UNIVERSIDAD CARLOS III DE MADRID

Trabajo de Fin de Grado Ingeniería Informática

Solución para la Digitalización de Museos y Espacios de Exposiciones mediante IOT

Autor:
Gonzalo Arévalo Planelles

NIA:
100083960

Tutores:
Jesús García Herrero
Álvaro Luis Bustamante

Titulación:
Doble Grado Ingeniería Informática y Administración y Dirección de Empresas

AGRADECIMIENTOS

A mis padres, mi hermano y mi familia, por apoyarme siempre y por motivarme en la idea de este proyecto.

A Sonia, por soportarnos tanto, al Trabajo de Fin de Grado como a mí, no una vez, si no dos veces, uno por cada carrera.

A Ali, por ser mi apoyo incondicional.

A todos mis amigos y amigas, por ser una de las vías de escape cuando era necesario.

A Marina, mi compañera de prácticas durante toda la carrera, y que en el único trabajo que no la he tenido al lado ha sido en este.

A mis tutores, por la ilusión que han mostrado siempre en este proyecto y la atención total que han tenido siempre conmigo.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN:

1.1. MOTIVACIÓN

Durante mis prácticas curriculares en **Oracle**, tuve la oportunidad de conocer más acerca de las tecnologías emergentes basadas en la **extracción de datos, el trato de los mismos** y conseguir el valor que las empresas necesitan. Formé parte del grupo de desarrollo de negocio de Big Data y apoyé logísticamente varios proyectos de **Internet Of Things**, conociendo las implicaciones técnicas y cuáles eran los activos de esas tecnologías que podían resultar atractivos para cualquier empresa.

De esta forma, puesto que en mi carrera es necesario presentar un trabajo de fin de grado para Ingeniería Informática y otro para Administración y Dirección de Empresas, pensé que explicar en el conjunto de los dos proyectos los beneficios de usar este tipo de tecnología podría ser muy interesante.

Así, durante varias sesiones con mi tutor de Informática, Jesús García, fijamos el trabajo en el desarrollo de una **solución** basada en Internet de las Cosas para el interior de un museo, de tal forma que cada objeto expuesto tuviera un beacon que se comunicara con un aplicación móvil, y, enviando la correspondiente consulta a un servidor, podemos interpretar qué era lo que tenía el usuario delante y ofrecerle un entorno interactivo que mejorara la **experiencia de usuario** y que permitiera al museo o espacio de exposiciones tener más información de cuál es el comportamiento de sus visitantes, qué obras son más atractivas y así poder mejorar los **flujos de movimiento** dentro del museo de la manera más adecuada y **maximizar las ventas** de aquellos souvenirs de las obras que más gusten a los usuarios.

Sobre todo, me resultó muy interesante poder extraer valor de esta solución: a través de ella, podíamos crear una fidelización de usuario a partir de la aplicación móvil, conocer los patrones de conducta dentro del museo, sus gustos, ofrecer una mejor experiencia de usuario, conseguir una segmentación de clientes que pueda favorecer un marketing estratégico más eficiente, o promociones en souvenirs en tiempo real que pudiera hacer que la venta de los mismos creciese.

Además, mi tutor me animó a que, aunque en un principio nos enfocáramos en el desarrollo de la solución para museo o exposiciones, pusiésemos énfasis en las ventajas de que este tipo de soluciones conforme estaba planteado pudiera ser extrapolable a otro tipo de negocios y que era totalmente móvil, aprovechando los datos explicados en la introducción sobre las bondades de los **smartphones**.

Posteriormente me reuní con mi tutora del trabajo de fin de grado de Administración y Dirección de Empresas, y tras debatir sobre este tipo de solución y sobre los distintos tipos de trabajo que podía

desarrollar, llegamos a la conclusión de que la mejor opción era desarrollar un **Business Plan** de una empresa que se dedicara al desarrollo y comercialización de este tipo de soluciones, dado que con todas las implicaciones empresariales del valor extraído del desarrollo y uso de esta solución, creíamos que era ideal para tener un plan de negocio con unos fundamentos sólidos.

Una vez entregado el Trabajo de Fin de Grado de Administración y Dirección de Empresas, y tener una gran acogida por parte de mi tutora y del tribunal que lo evaluó, comencé con el desarrollo completo de la solución para conseguir un producto que cumpliera con las características que se han mencionado anteriormente, y demostrar así su viabilidad.

De esta forma, tras la creación del plan de empresa de Lighthouse Solutions, empresa que se encargará de desarrollar toda esta actividad, comencé con el desarrollo de la solución.

1.2. OBJETIVO

1.2.1. INTRODUCCIÓN

La situación actual del uso de tecnologías móviles de forma masiva y la necesidad de las empresas de aprovechar todo ese flujo de información supone una oportunidad para ayudarlas a adaptarse a este cambio.

El objetivo es crear una **solución adaptada a negocios** que tengan una sede física, a la que necesiten atraer usuarios y hacer que la experiencia de usuario en su interior mejore, y a su vez se obtengan rendimientos crecientes de esta actividad, explotando la fidelización del cliente, la mejora de la experiencia de usuario y que puedan extraer conclusiones del comportamiento de sus clientes en el interior de su establecimiento. Por tanto, será una solución innovadora. En un primer momento, Lighthouse Solutions comercializará la solución para museos y exposiciones, bajo el nombre de **Flares**, pero en el futuro la intención es que este tipo de solución sea extrapolable a cualquier negocio con sede física y se pueda extraer valor del **comportamiento del cliente** con la aplicación del móvil del negocio y sus patrones de **comportamiento en el interior** del espacio físico.

Así, la solución estará basada en el Internet de las Cosas, es decir, en dotar a los objetos de acceso a internet y en capacidad de comunicarse con otros dispositivos. En nuestro caso, cada objeto dispondrá de un **beacon**, que es un dispositivo que se comunica con otros mediante **bluetooth**, y que actuará como un «faro», diciendo únicamente cuál es ese beacon y donde está. El dispositivo con el que se comunican estos beacons es una aplicación móvil, que en función de lo cerca que este de uno de los beacons,

significará que se encuentra cerca del objeto al que se ha añadido el dispositivo. A partir de aquí, se abren un sin fín de posibilidades, desde el uso tradicional de una aplicación móvil para **generar demanda**, enviando ofertas o información a los usuarios de la aplicación, hasta mostrar **contenido interactivo** en función de la localización del usuario y de su cercanía con un objeto del negocio determinado, de tal forma que el usuario también pueda elegir qué contenido ver, y pueda opinar de aquello que se le muestra, obteniendo así un perfilado del usuario. Además, se desarrollará un back-end en el que se tratará toda la **inteligencia de negocio**, y se aprenderán los patrones de movimiento dentro del local, y toda la información u oferta que se muestre a los usuarios se hará con el aprendizaje histórico de usuarios del mismo perfil y que estadísticamente puede interesarle.

El mercado al que va dirigida la actividad de Lighthouse Solutions es el de organizaciones que basan su actividad en la visita de sus posibles clientes a un espacio propio, ya sea un local o una sede, para ofrecerle algún tipo de venta de productos o exposición de los mismos. Por tanto, principalmente, nuestra actividad irá dirigida a museos u organizaciones que se encarguen de exposiciones, que necesiten una gestión del espacio, una generación de demanda a partir de una aplicación móvil, dándoles la oportunidad de extraer una inteligencia de negocio de la interacción del usuario con la aplicación al visualizar la información de los eventos, mientras que asiste a uno y valora la exposición en general o algún objeto expuesto en particular, y de cómo realizar una gestión eficiente de los espacios en función de los gustos de usuario y sus patrones de conducta en el interior.

1.2.2. PRODUCTO

En cuanto a los objetivos meramente técnicos del trabajo, estos están relacionados con la carrera de ingeniería informática y más concretamente con la rama de computación. Dentro de esta, al tratarse de una solución que engloba en sí misma varios productos, tenemos Base de Datos, con la intención de organizar, almacenar y recuperar grandes cantidades de datos, Ingeniería de Software, como estudio de diseño, implementación y modificación de software (desarrollo de **API-Rest y aplicación móvil**) y, en menor medida, Ciencia de la Información por el enfoque de extraer conclusiones a través de datos objetivos que provienen de distintos inputs.

El primer objetivo es el análisis de las capacidades de este tipo de soluciones que en futuro deberían suponer un gran impacto para las empresas: a través de la combinación de distintos productos con unas características determinadas, probar una solución que sea modular y exportable a distintas necesidades.

El siguiente objetivo, en el **front-end** es la configuración y desarrollo de una aplicación móvil **iOS** desarrollada en lenguaje Swift, con **beacons** de marca Estimote. En el **back-end**, la configuración de un api-rest con una base de datos **No-SQL** (cuyos motivos de elección se detallarán más adelante),

tecnología que se encuentra ahora mismo muy de moda por las capacidades que tiene para funcionar con gran cantidad de datos con buenos rendimientos con paradigmas que de aquí en adelante deberían dar que hablar, como son el **IOT** y el **Big Data**.

El objetivo final es el desarrollo de una solución capaz de digitalizar espacios de exposición y museos, como conjunto de funcionamiento de cada uno de los elementos que conforman la solución.

También es un objetivo la creación de una aplicación móvil de gestión que permita al administrador del museo gestionar la solución en función de las necesidades. Es decir, no me centro exclusivamente en el desarrollo de la aplicación de usuario si no de las funcionalidades de gestión y extracción de información relevante que sea de interés para el museo como entidad, y no solo desde el lado de usuario.

1.3. MEDIOS

Los medios utilizados para el desarrollo de este proyecto han sido un ordenador, dos teléfonos con sistema operativo iOS, y 6 beacons de marca Estimote. Todos los equipos han sido puestos a disposición por mi persona, ya sea poniendo equipos de mi posesión o adquiriéndolos para el desarrollo del proyecto. A continuación, se describen las características de cada uno de ellos:

Ordenador:

Marca	MacBook Pro (Retina 13 pulgadas)
Procesador	2,7 GHz Intel Core i5
Memoria Ram	8GB 1867 MHz DDR3
Disco Duro	250 GB
Sistema Operativo	macOS Sierra

Tabla 1- Medios: Computadora

Coste aproximado del equipo: 1500 €

Ilustración 1 - Computadora utilizada

Dispositivos móviles:

Marca	iPhone 6s 16GB 4,7”
Procesador	Chip A9 (coprocesador M9) con arquitectura de 64 bits
Memoria Ram	2 GB
Disco Duro	16 GB
Sistema Operativo	iOS 10

Tabla 2 - Tabla de dispositivo móvil (I)

Marca	**iPhone 7 Plus 128GB 5,5”**
Procesador	Chip A10 (coprocesador M10) con arquitectura de 64 bits
Memoria Ram	3 GB
Disco Duro	128 GB
Sistema Operativo	iOS 10

Tabla 3 - Tabla de dispositivo móvil (II)

Ilustración 2 - Dispositivos móviles utilizados

Coste aproximado de ambos equipos: 550 € + 1.100 €.

Beacons:

Marca	Estimote
Procesador	32-bit ARM Cortex MO CPU
Memoria Ram	256 KB

Tabla 4 - Tabla de beacons utilizados

Coste aproximado del equipo: 178 €

Ilustración 3 - Beacons Utilizados

1.4. ESTRUCTURA DEL DOCUMENTO

En este apartado, se va a detallar el contenido de cada una de las partes que integran este documento, a fin de entender mejor en qué consiste este documento y facilitar su comprensión. El documento consta de seis apartados, tratándose en cada uno de ellos los siguientes temas:

1.4.1. INTRODUCCIÓN

Este apartado, al que pertenece el subapartado en el que se encuentra, sitúa al lector en un **contexto general** a fin de entender a grandes rasgos en qué consiste el trabajo, cuales son los objetivos que se buscan con el desarrollo del mismo, y la información que contiene el documento.

1.4.2. ESTADO DEL ARTE

En este apartado se explicará el **contexto tecnológico** en el que se sitúa la organización de la cual se quiere desarrollar el plan de empresa. Primero, se explican los factores tecnológicos que impulsen el análisis de todos los apartados para concluir si sería viable este negocio, haciendo también un estudio similar del contexto sobre la tecnología que se va a utilizar y productos similares que pudiera haber en el mercado.

1.4.3. PRODUCTOS Y SOLUCIÓN

En este apartado se detallará todo lo relacionado con cada uno de los productos a desarrollar por separado, y la integración de unos con otros hasta dar con la solución buscada. Es un apartado técnico. Consta de seis subapartados. En el primer apartado, se abordará una explicación introductoria que de manera general expondrá las partes que componen la **solución** y la **configuración** que se lleva a cabo de cada uno de ellos.

En el siguiente apartado, se explica en detalle el algoritmo empleado para conseguir la **funcionalidad** requerida de cada uno de los elementos que conforman la solución. Este será el subapartado 3.2.

El subapartado 3.3 se obtendrán las especificaciones del sistema, recopilando todo lo que debe hacer este. Posteriormente, en el siguiente subapartado, se tratará la forma en que se resuelve cada uno de los problemas planteados durante el análisis, detallando la **arquitectura del sistema** y sus distintos componentes.

En el subapartado 3.5 contendrá una planificación y presupuesto estimados del proyecto software realizado. Por último, se añadirá un manual que indica cómo utilizar el software por parte del administrador del sistema, y por parte de cualquier usuario final que sea visitante del museo.

1.4.4. CONCLUSIONES

En este apartado se analizará si los objetivos planteados en la introducción del trabajo finalmente se han cumplido y en qué medida, así como las experiencias y el aprendizaje personal que se ha tenido a lo largo de la realización del proyecto.

1.4.5. BIBLIOGRAFÍA

Se incluirá un apartado para detallar la bibliografía que se ha consultado para la realización del trabajo para los distintos apartados, ya que a lo largo del trabajo se puede hacer referencia a alguno de los documentos.

1.4.6. ANEXO

En este apartado se incluirá todo aquello que no forma parte de la estructura troncal del trabajo, pero si lo complementan o son necesarios para la realización de alguna de las partes, o bien arrojan más detalle de alguna de ellas.

2. ESTADO DEL ARTE.

2.1. CONTEXTO TECNOLÓGICO GENERAL

A día de hoy, en pleno año 2017, la conectividad a gran escala con el resto del mundo es un hecho. Y si hubiera algún dispositivo en el que resumir esto, sería el **smartphone**.

Los «antiguos» teléfonos móviles nos permitían hacer llamadas de voz desde cualquier lugar (que tuviera una antena cerca) y con un factor que lo hacía especial: la movilidad. Esto supuso un cambio en la forma de comunicarse entre los usuarios, ya fuera mediante comunicación por voz o por mensaje de texto, pero desde cualquier lugar.

Pero la llegada del smartphone supone una vuelta de tuerca más. Además de las prestaciones de sus antecesores, añade la posibilidad de añadir un sin fin de prestaciones gracias a la conexión a internet de la que gozan este tipo de dispositivos. Hoy en día, cualquiera que disponga de un smartphone en su poder, puede realizar tareas tan sencillas como consultar el correo o el estado del tiempo o las carreteras en **tiempo real**, tareas algo más elaboradas como comprar productos desde el móvil, o simplemente estar en contacto con otras personas mediante el uso de redes sociales.

De hecho, ya existen en el mundo más número de smartphones que de personas (7.100 millones vs 7.900 millones), y más de la mitad de las visitas que reciben los grandes buscadores proceden ya del móvil, siendo un 62% el tiempo total pasado por los usuarios en el mundo online desde smartphones y tablets.

En cuanto a España, lidera el ranking de penetración de smartphone en la población en Europa meridional, y es el país con más smartphone por población del mundo, que, en los últimos años, ha pasado del 63% al 81% y 87% respectivamente. El 41% de la población accede a las redes sociales desde el móvil, por encima de la media global del 27%. Los usuarios de entre 18 y 34 años dedican una media de 29.6 horas al mes en redes sociales, mientras que los mayores de 55 18,3 horas. No obstante, los españoles lo utilizan principalmente para acceder al correo electrónico (87%) y navegar por la red.[18]

Como es de suponer, el flujo de información en tiempo real que se produce en todo el mundo es enorme. A continuación, se muestra una aproximación de los datos generados únicamente durante un minuto en

tiempo real, teniendo en cuenta las plataformas más utilizadas a nivel mundial y con las que podemos extrapolar para lo que usan los usuarios internet. [17]

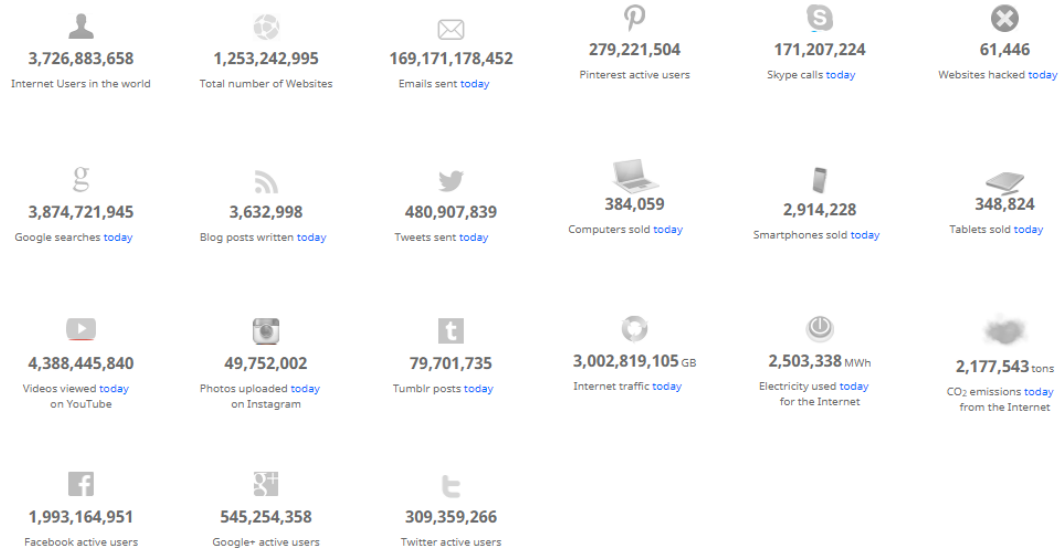


Ilustración 4 - Internet en Tiempo Real

A partir de la imagen anterior, podemos hacernos una idea de la cantidad de datos que se pueden generar a lo largo del tiempo. De hecho, en el tercer elemento de la cuarto columna podemos ver que, solamente en lo que llevamos del día en que se escriben estas líneas, se han generado **3.002.819.105 Gigas** de tráfico de información. Y quedan aún 7 horas para que se acabe el día. En el resto de la imagen podemos observar a grandes rasgos qué tipo de información se ha generado y transmitido en función de las plataformas más utilizadas por los usuarios.

Pero, ¿y si las empresas usaran esta información para **maximizar sus ganancias**? ¿Y si lo utilizaran para **minimizar sus costes**? ¿O para **generar más demanda**? ¿O para conocer más acerca de los usuarios y tener un **trato** con ellos mucho más eficiente **en función de sus gustos**?

Muchas empresas se han dado cuenta de la ventaja que supone conocer los gustos de usuario, sus patrones de actuación, y poder hacer una **segmentación** de los mismos de forma más eficaz, para obtener ventaja en el análisis de los datos y emplearlos en una inteligencia de negocio más eficiente y que ayude en mayor medida al desarrollo del negocio. De hecho, el 86% de los directores de Marketing considera que sus campañas a través de móviles son efectivas o muy efectivas.

Por todos estos datos, para las empresas se convierte en una obligación el hecho de estar en internet, y en facilitar el acceso a sus servicios a través de los smartphones, encontrándose en plena transición para

sacar partido a todo este conocimiento embebido en los datos generados masivamente, y que hay que explotar.

Además, según la empresa Daemon Quest Deloitte, la linea de consultoría de Deloitte especializada en estrategia de clientes, marketing y ventas, afirma que los usuarios están demandando un trato especializado, y que solo aquellas empresas que sepan adaptarse a este hecho y situar al cliente en el centro de su estrategia de negocio, tendrán garantizado el éxito. Daemon Quest Deloitte, divide este giro organizativo hacia el cliente en tres ramas: estrategia de crecimiento basado en **digitalización** y **tecnología**, **customer experience, y fidelización**.

A partir de ahora, las empresas necesitarán soluciones que, a través de la tecnología de internet móvil, pongan a disposición del usuario una **información ampliada** de sus productos o servicios, un **trato personalizado** para cada usuario, aprovechar la movilidad de los usuarios (también en interiores), interactividad y vinculación, para mejorar así la experiencia de usuario y que este se sienta bien al usar nuestros servicios para que repitan, consiguiendo una **fidelización**.

2.2. PRODUCTO

2.2.1. SMARTPHONES

Como se ha explicado en el punto anterior, la llegada de la tecnología inteligente a los teléfonos celulares o móviles tradicionales, revolucionó por completo la funcionalidad de los mismos.

Además de la posibilidad de llamada o mensaje de texto desde cualquier lugar, el poder acceder a redes sociales, visitar páginas de nuestro interés, o usar según qué aplicaciones, nos hace tener en nuestro poder una ventana que nos conecta con el mundo. Y no solo eso. También decimos mucho de **qué nos gusta y qué nos interesa más en cada momento**.

La posesión de un Smartphone tiene un valor crítico tanto para sus poseedores como para aquellos que les pueda interesar estar en contacto con nosotros. Como se ha dicho anteriormente, es una **ventana de conexión**, pero en dos sentidos. Para las empresas se ha vuelto vital estar disponibles desde cualquier lugar, teniendo una aplicación móvil o una web, a través de la cual puedan aprender de los clientes y usuarios, aumentar su experiencia de usuario, generar demanda, realizar ofertas…

Ser **accesible desde un Smartphone** se ha convertido en una palanca fundamental de muchos negocios para incentivar su actividad. De hecho, no es raro las empresas que no disponen de sede física pero sí de sede electrónica, ya sea vía web, por aplicación móvil, o por ambas.

2.2.2. INTERNET DE LAS COSAS

Por otro lado, a fin de contextualizar este proyecto, se debe explicar una parte fundamental del mismo: Internet de las Cosas (IoT, iniciales de Internet of Things). Consiste en un **paradigma** que ha tomado mucha relevancia en los últimos tiempos, y que sostiene que, de la misma forma que la llegada de internet produjo un cambio sustancial en la vida cotidiana tanto de personas como empresas, no se va a quedar así, y seguirá avanzando hacia un modelo en el que todo tenga una conexión a internet.

Y es que cuando se dice todo, es todo. La idea de IoT sostiene que **cualquier objeto**, llegado el momento, **será dotado de la capacidad de tener acceso a internet**, y al actuar como nodos de la red de redes, poder interactuar unos con otros. Cada uno de los objetos tendrá un identificador único y unívoco, que le permitirá ser diferenciado de cualquier otro. No necesariamente debe ser un objeto como imaginamos en nuestra cabeza. También determinados sensores pueden formar parte de esta red y al interactuar entre ellos, obtener un determinado resultado. Por ejemplo, imaginemos que la caldera de nuestro domicilio, así como los radiadores, constan de un dispositivo controlador que, además, le proporciona acceso a internet. Los dispositivos de los radiadores se comunican con un sensor de temperatura que hay en su misma habitación, y además, puede comunicarse con el dispositivo de la caldera. Dependiendo de la temperatura que haya en cada habitación, el dispositivo del radiador que se encuentra en el mismo habitáculo cerrará o abrirá su llave de paso en función de la temperatura deseada. Esto, que no es nuevo, ya que es un controlador electrónico sin más, si le añadimos el factor de la conexión a internet, da la posibilidad, por ejemplo, que desde nuestro trabajo podamos elegir la temperatura a la que queremos que se encuentre el domicilio a nuestra llegada, y cuando apagarlo o encenderlo de manera remota. O elegir qué habitaciones queremos que estén más calientes que otras si en casa hay un niño pequeño, o si el resto de la familia está de viaje, únicamente encender la calefacción del dormitorio y la sala de estar.

Como podemos observar, las posibilidades de este nuevo paradigma son casi **infinitas**, hasta donde nos alcance la imaginación. Además, IoT supone una nueva **fuente de entrada de datos** para análisis de cualquier objeto o conjunto de estos, ahorro energético, o monitorización en remoto de, incluso, entornos de objetos más grandes y complejos.
Se debe tener en cuenta, además, que esta nueva tecnología camina de la mano de otras dos más grandes que están destinadas a revolucionar los datos tal y como los conocemos: Big Data y Cloud. IoT junto con Big Data supone una nueva entrada de datos que, en combinación con otros y análisis de los

mismos, puede suponer la extracción de un valor diferencial al que antes no teníamos acceso. Además, podríamos pensar que toda esta nueva fuente de datos hay que guardarla en algún sitio y, que, si aumenta en exceso, puede ser caro. Con Cloud, este problema se alivia, ya que el almacenamiento en la nube será rápido y barato. Incluso Big Data en un Cloud Computing puede combinar toda esta tecnología en una sola plataforma de manera eficiente.

Por último, se espera que los beneficios de tener un sistema IoT sean tales, que no se limitaría únicamente a hogares y empresas, si no que llegarían a existir también **Smart Cities** en las que semáforos, papeleras, contenedores, sensores de contaminación, etc. podrían ayudar a que una ciudad fuera más eficiente gracias a la información generada por los sensores y los habitantes de la misma.

2.2.3. PRODUCTOS SIMILARES

En cuanto a los productos similares, debemos tener en cuenta que este no es un producto como tal, sino una solución que consta de varias partes que, combinadas entre sí, resuelven un problema de negocio.

Como hemos explicado en puntos anteriores, las partes principales, sin llegar a un nivel de tecnología muy bajo, es la aplicación y los dispositivos bluetooth. Si bien muchos museos u organizaciones de exposiciones ya constan de aplicación móvil, únicamente **cubren la parte de generación de demanda**, de marketing o publicidad, como una forma de conectar con el cliente. Pero no lo utilizan para **conocer** más **al cliente**, ni para ofrecerle un **contenido interactivo**, o **más información** de la exposición que estén visitando. Tampoco utilizan los datos que generan los usuarios para hacer un **estudio de los patrones de movimiento** dentro del espacio de exposición.

Si bien hay empresas que se dedican al desarrollo de este tipo de soluciones, como Gennion Solutions, no tienen un desarrollo especializado para este tipo de espacios. Aunque se adaptan a las peticiones del cliente, la posibilidad de centrarnos en ese público objetivo con una solución que se adapta perfectamente a las necesidades de estos espacios y hacerlo de forma específica puede reportar notoriedad sobre los posibles compradores.

En los últimos años, ha aparecido una empresa nueva, Seeketing que desarrolla este tipo de solución sin la necesidad de que sea mediante un dispositivo bluetooth ni siendo necesario la posesión de una aplicación móvil para recibir notificaciones. En mi opinión, esta práctica es demasiado invasiva, pues se hace mediante mensajes de texto tradiciones por la frecuencia que provoca la red móvil del teléfono. Aunque así se pueden estudiar triangulación de teléfonos mediante las antenas y se sigue pudiendo enviar información, seguramente esta solución está enfocada a otro tipo de negocio. Para un espacio de

arte y cultura, el **proveer información, el fidelizar al cliente** sobre tu espacio y que pueda **interactuar con tu aplicación** solo si él lo desea, es fundamental para este tipo de mercado.

Por tanto, con este proyecto no se pretende realizar una solución totalmente novedosa, que también, sino adaptarlo a unas necesidades determinadas y explotar ese mercado al máximo.

2.3. TECNOLOGÍA

A continuación, con el fin de explicar la elección de cada uno de los elementos que conforman la solución completa, se realizará una breve explicación con un gráfico para entender qué pretendemos conseguir con cada una de las partes y por qué se ha elegido cada hardware o software.

2.3.1. BREVE EXPLICACIÓN DE ELEMENTOS QUE CONFORMAN LA SOLUCIÓN

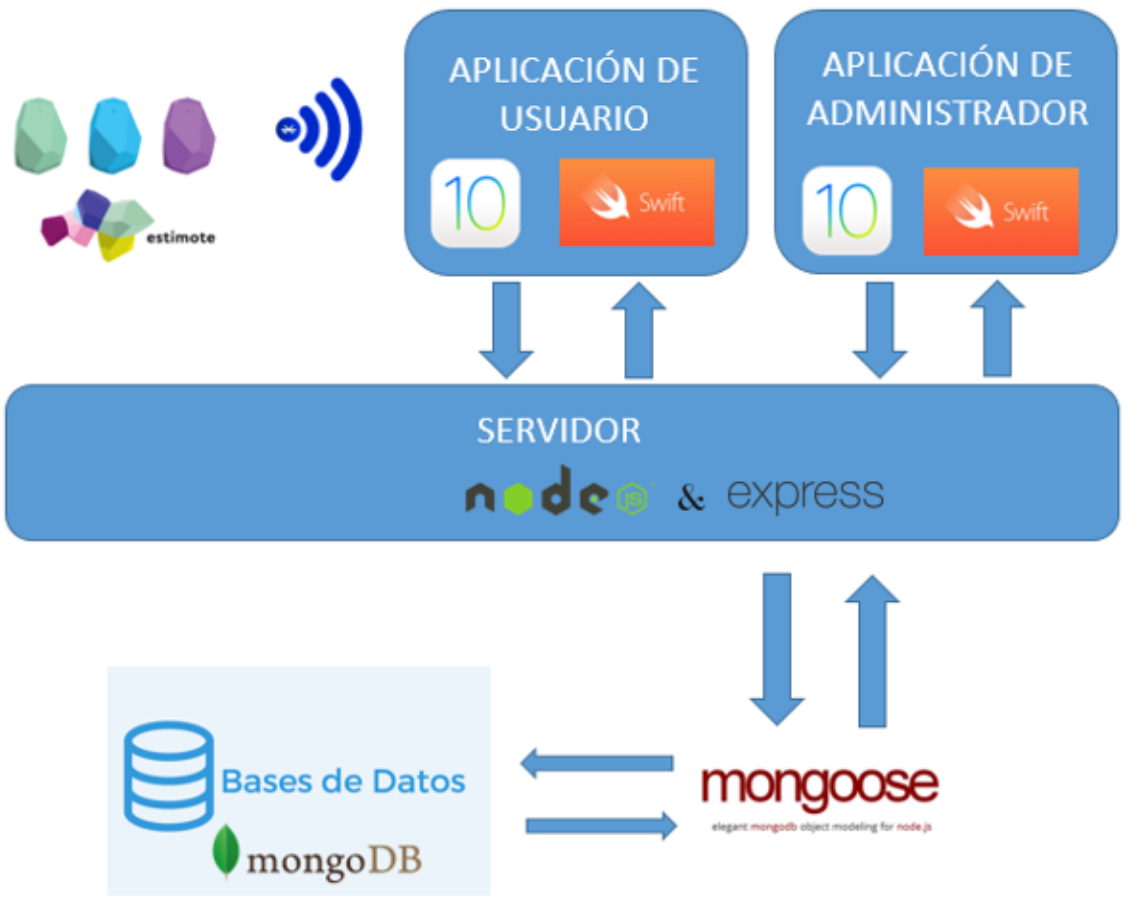


Ilustración 5 - Arquitectura del Sistema

Antes de explicar el motivo por el que se ha elegido cada uno de estos componentes y no otros que se encuentran actualmente en el mercado, vamos a realizar una pequeña exposición de cómo funcionaría la solución propuesta, con el fin de entender cada elemento que la compone.

Como hemos explicado anteriormente en el presente documento, la solución pretende proveer al museo de la capacidad de fidelizar al usuario, conocer sus patrones de comportamiento en el interior del mismo, proporcionarle un entorno interactivo que le sea atractivo y le proporcione valor, y que todo ello repercuta en una mejor gestión del museo o del espacio de exposiciones, haciendo que los visitantes vuelvan, mejorando la distribución de obras en el interior del museo y maximizando las ventas de la tienda de souvenirs sabiendo qué ofrecer a qué usuario en función de sus gustos y su comportamiento. A fin de cuentas, una solución que permita la digitalización de este espacio mediante Internet de las Cosas.

Para ello, el usuario podrá descargarse una aplicación móvil, en la que podrá **registrarse** y **autenticarse** para aprovecharse de la información que le proporciona e interactuar durante su visita. Esta aplicación **se comunicará con los beacons**, que son dispositivos bluetooth que lanzan una señal unívoca por la que podemos identificar qué beacon es. La aplicación, a través de la intensidad de señal que le llega de cada beacon, sabrá cuál de ellos es el que tiene más cerca. Teniendo esta información, lanzará una petición al servidor para que le devuelva la información relativa a la obra que tiene más cercana y que va vinculada al mencionado dispositivo bluetooth.

El **servidor**, una vez reciba esta petición, lanzará una consulta a la base de datos para saber qué información está vinculada al **minor number** del beacon que tiene más cercano. Una vez disponga de esta información, la enviará a la aplicación móvil en formato **JSON**. La aplicación móvil permitirá puntuar la obra para conocer cuál es el gusto de este usuario, además de guardar el tiempo que ha estado delante de esta obra observándola. Al cerrar sesión, el usuario, si cumple con unas reglas preestablecidas de inteligencia de negocio, recibirá un vale descuento de aquella obra que haya evaluado mejor (y observado durante más tiempo).

Existirá otra aplicación, destinada al **administrador del sistema**, que le permitirá vincular un nuevo beacon a una **nueva obra, actualizarla, consultarla o borrar** esta vinculación. Además, podrá ver un apartado de **estadísticas** en el que gráficamente se muestra en cada obra cual es la siguiente a la que pasan los usuarios, de tal forma que se puede extraer el **flujo de visitantes** a través del museo. También se puede saber el **tiempo medio que un usuario está delante de un cuadro**. Así, podemos saber qué obras deben contar con un espacio más amplio de exposición, dada la posible aglomeración de visitantes alrededor del mismo mientras lo observan durante largo tiempo.

En cuanto al servidor, cuenta con los **modelos de usuario y de obras**, con los datos que debemos almacenar y tener en cuenta para el correcto funcionamiento. Contiene todas las rutas de las peticiones, así como las funciones que debe desempeñar para el correcto funcionamiento de la aplicación.

En cuanto a la base de datos, es de tipo **NoSQL**, y almacena la información de usuario y de las obras expuestas. El porqué de este tipo de base de datos, así como de la distribución o lenguaje utilizado en el resto de elementos de la solución, será explicada a continuación.

2.3.2. LOCALIZACIÓN INTERIOR: BEACONS

Ante el planteamiento de la solución que se quería dar para el problema de la digitalización de un espacio de exposición o museo, se precisaba conocer qué tipos de localizaciones internas hay actualmente en el mercado, analizarla, y decidir cual se adapta mejor a la resolución de nuestra problemática. A continuación, se muestra un cuadro comparativo de los métodos para localización interna más generalizados.

	GEOREFERENCIA	**WIFI**	**BEACON**
PRECISIÓN Y USO	Exterior	Interior	Interior más localizado
APLICACIONES DE MARKETING	Entregar mensajes móviles para conducir el tráfico a pie cercano, generar percepciones de comportamiento	Ofrecer contenido móvil general (sin aplicaciones) o experiencias móviles personalizadas (con aplicaciones	Proporcionar contenido y experiencias móviles personalizadas en un lugar específico del interior.
CAPACIDAD DE MEDICIÓN	Tráfico a pie en el vecindario y fidelización móvil.	Frecuencia de visita, duración y participación móvil.	Verificación de la frecuencia localizada, duración y participación móvil, interacciones de contenido.
APLICACIÓN MÓVIL REQUERIDA	Si	No	Si, en la mayoría de los casos, aunque no es necesario para Google Nearby Notifications.
CONEXIÓN BLUETOOTH REQUERIDA	No	No	Sí

TRIGGER DEL CONSUMIDOR	Posibilidad de notificaciones emergentes automáticas, o al abrir el usuario la aplicación.	Al abrir el usuario la aplicación o el navegador.	Posibilidad de notificaciones emergentes automáticas, cuando el usuario abra la aplicación.
PÚBLICO OBJETIVO	Aplicaciones para empresas minoristas	Empresas minoristas, aplicaciones de terceros, navegador web móvil.	Empresas minoristas, aplicaciones de terceros.

Tabla 5 - Comparativo de sistemas de localización

Analizando lo tratado en el cuadro anterior, podríamos concluir que cualquier de los tres métodos nos ayudaría a dar solución al problema planteado. No obstante, debemos analizar también, cual es el que más se adapta a nuestras necesidades, tanto en aspecto técnico como de coste material, coste efectivo, desarrollo, etc.

Por tanto, en primer lugar, podemos descartar la georreferenciación. Hoy en día todos nuestros smartphones están capacitados de una antena GPS, por lo que sería posible utilizarlo para nuestra aplicación. No obstante, sería un desperdicio de costes y usar esta tecnología para una localización de unos pocos metros cuadrados, cuando la georreferenciación está destinado a grandes cantidades de terreno.

De esta forma, el planteamiento debía ser si usar tecnología Wifi o beacon. Aunque ambas tecnologías están destinadas al interior, el beacon se adaptaría en este caso mejor en cuanto a una localización mucho más **específica**. Además, el coste efectivo de desarrollo de un sistema mediante Wifi, en el que hay que medir señales, hacer triangulaciones, y crear un mapeo dentro del lugar donde se instala la solución, en contraposición con los beacons, que tiene un coste efectivo de desarrollo menor, y no hace falta mapear, si no que puede ser totalmente **modular y adaptable** a distintas localizaciones independientemente de su forma, es un punto más a favor para la tecnología beacon.

Un aspecto a tener en cuenta, es que la intención del desarrollo de esta aplicación, seguido con la modularidad y adaptabilidad, es que el propio administrador de esta tecnología en el museo pueda realizar los cambios necesarios de exposiciones, ampliar beacons, borrarlos su conexión con una obra, etc. sin la necesidad de recurrir, al menos en casos triviales para el administrador del museo, de los desarrolladores.

En cuanto a todos aquellos aspectos relacionados con la necesidad o no de usar una aplicación móvil o interactuar con el usuario de forma directa mediante notificaciones no pactadas, aunque estas puedan ser de tremendo interés para el usuario, he decidido que todas las ventajas que da la solución sean bajo demanda directa del usuario, otorgándole un entorno personalizado e interactivo, pero únicamente cuando él lo solicite. Como hemos analizado en el punto 2.2.1 sobre smartphones, está debidamente justificado que el usuario por sí solo usa aplicaciones en su móvil y que demanda este tipo de servicios, por lo que no es necesario intentar distribuirlos sin ser bajo demanda directa.
Además, consultando marcas de beacons, nos da la posibilidad de ampliar modularmente servicios para la digitalización de museos y espacios de exposición e interiores en general, ya que muchos de ellos vienen equipados con sensores de temperatura, humedad, etc. Aunque estos asuntos no están estrictamente relacionados con el proyecto actual, si podría estarlos con posibles trabajos futuros.

Por tanto, por versatilidad, por adaptación a la problemática, por eliminar complejidad en el desarrollo y en el uso por parte del administrador del museo, creemos que la opción más adecuada es la tecnología beacon.

Una vez decidida la tecnología a utilizar, debemos elegir entre la distinta oferta de beacons que hay actualmente en el mercado:

Modelo	iBeacon Bluetooth Low Energy 4.0 BLE
Marca	Avvel International
Rango	40 m
Bluetooth	4.0
Memoria Ram	16 Kb
SDK	Android e iOS
Autonomía	12 meses
Tamaño	27 x 7,6 mm
Peso	15 g
Otras Especificaciones	Mala valoración de usuarios y falta documentación de uso
Coste	16,50 €/ud

Tabla 6 - Características beacon (I)

Ilustración 6 - Beacon propuesto (I)

Modelo	iBeacon ibec001
Marca	Social Retail
Rango	45 a 82 m
Bluetooth	4.0
Memoria Ram	No especifica
SDK	Propia, para cualquier app
Autonomía	18 meses
Tamaño	38 x 38 x 6 mm
Peso	41 g
Otras Especificaciones	Sin carcasa, mala valoración de usuarios, sin batería de Litio
Coste	16,99 €/ud

Tabla 7 - Características beacon (II)

Ilustración 7 - Beacon propuesto (II)

Modelo	iBeacon i8
Marca	Social Retail
Rango	90 m
Bluetooth	4.0
Memoria Ram	No especifica
SDK	Propia, para cualquier app
Autonomía	18 meses
Tamaño	43,6 x 43,6 x 21,3
Peso	181 g
Otras Especificaciones	Carcasa de silicona, valoración aceptable de usuarios, requiere plataforma de pago para uso y activación.
Coste	28,99 €/ud

Tabla 8 - Características Beacon (III)

Ilustración 8 - Beacon propuesto (III)

Modelo	iBeacon Bluetooth Low Energy 4.0 BLE (Largo Alcance)
Marca	Avvel International
Rango	75 m
Bluetooth	4.0
Memoria Ram	No especifica
SDK	Android e iOS gratuitos, App Android de configuración rápida
Autonomía	12 meses
Tamaño	94 x 94 x 18 mm
Peso	18 g
Otras Especificaciones	Poca precisión al medir la distancia.
Coste	21,00 €/ud

Tabla 9 - Características beacon (IV)

Ilustración 9 - Beacon propuesto (IV)

Modelo	Proximity Beacons
Marca	Estimote
Rango	70 m
Bluetooth	4.2 LE Standard
Especificaciones técnicas	ARM Cortex M4 – procesador 32 bit con FPU, 64Mhz, 512 kb Flash, 64Kb Ram.
SDK	Android e iOS gratuitos, app y web de configuración rápida
Autonomía	24 meses
Tamaño	55 x 38 x 18 mm
Peso	30 g
Otras Especificaciones	Incorpora NFC, sensor de movimiento (acelerómetro), sensor temperatura.
Coste	16,50 €/ud (Se compra en paquetes de 3 beacons)

Tabla 10 - Característica beacon (V)

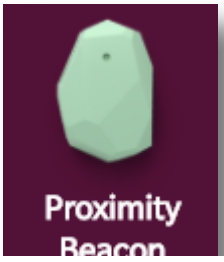


Ilustración 10 - Beacon propuesto (V)

Modelo	Location Beacons
Marca	Estimote
Rango	200 m
Bluetooth	4.2 LE Standard
Especificaciones técnicas	ARM Cortex M4 – procesador 32 bit con FPU, 64Mhz, 512 kb Flash, 64Kb Ram.
SDK	Android e iOS gratuitos, app y web de configuración rápida
Autonomía	60 meses
Tamaño	62 x 41 x 23 mm
Peso	67 g
Otras Especificaciones	Incorpora NFC y GPIO, sensor de movimiento (acelerómetro), sensor temperatura, sensor de luz ambiental, sensor de presión.
Coste	27,50 €/ud (Se compra en paquetes de 3 beacons)

Tabla 11 - Características Beacon (VI)

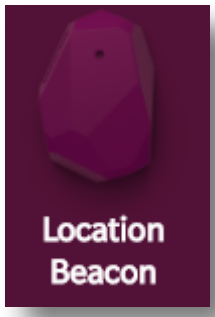


Ilustración 11 - Beacon propuesto (VI)

Modelo	Stickers Beacons
Marca	Estimote
Rango	7 m
Bluetooth	4.2 LE Standard
Especificaciones técnicas	ARM Cortex M0- 32 bit, 16Mhz, 256 kb Flash, 16Kb Ram.
SDK	Android e iOS gratuitos, app y web de configuración rápida
Autonomía	12 meses
Tamaño	62 x 41 x 23 mm
Peso	67 g
Otras Especificaciones	Incorpora NFC y GPIO, sensor de movimiento (acelerómetro)
Coste	8,25 €/ud (Paquetes de 10 Stickers)

Tabla 12- Características Beacon (VII)

Ilustración 12 - Beacon propuesto (VII)

Como hemos podido observar en las especificaciones técnicas recogidas en las tablas anteriores, los precios están bastante equiparados entre las distintas marcas. No obstante, el hecho de querer realizar una solución que sea **modular**, y con la **posibilidad de añadir funcionalidades** en el futuro que permitan avanzar en el desarrollo de soluciones IoT para interiores (como podría ser, por ejemplo, controlar también la temperatura del interior del museo para determinadas obras, o estancias), lo ideal es elegir los beacons de marca Estimote. Además, cuentan con una amplia documentación, un servicio de venta y postventa de gran calidad y una web de desarrolladores en la que, a modo de foro, se pueden consultar trabajos nuevos realizados, preguntar dudas, etc. con una comunidad de más de 100.000 desarrolladores. Además, tiene una amplia gama de productos que nos puede ayudar a elegir aquellos beacons que se adaptan mejor a nuestras necesidades. Por ejemplo, usar en el futuro Location Beacons para saber cómo es el tránsito en plano de los usuarios, ver si la luz en la sala es la correcta, usar los Proximity Beacons para la aplicación que vamos a desarrollar, y Sticker Beacon para lugares menos amplios, como por ejemplo, la tienda de souvenirs. Aunque para la realización del proyecto he adquirido seis unidades de Proximity Beacon, creo que tener en cuenta las posibilidades que proporciona esta marca ha de tenerse en cuenta para la elección del beacon utilizado.

Ilustración 13 - Desglose de beacon de Estimote

2.3.3. APLICACIÓN MÓVIL: IOS

Una vez seleccionados los beacons que se utilizarán para la realización del proyecto, debemos elegir si la aplicación móvil que se comunicará con estos será Android o iOS, puesto que ocupan el 99.6% del mercado de smartphones y, por tanto, las ideales para llegar a un público mayor.

En cuanto a las características de los beacons, en cuanto a la capacidad de trabajar con ambas, Estimote cuenta con **SDK** tanto para iOS como para Android. Por tanto, la elección debe realizarse en base a otros factores.

En mi caso, tanto en el trabajo como mi Smartphone personal, son iPhone. Además, cuento con ordenador personal Apple, MacBook Pro, y una computadora HP para el trabajo. Dado que tengo los medios necesarios para el desarrollo y pruebas en un dispositivo real para el desarrollo en iOS, y que durante la carrera del Doble Grado de Ingeniería Informática y ADE, en la parte de Informática y concretamente en la asignatura de dispositivos móviles habíamos trabajado con Android, consideraba que sería enriquecedor aprender también a programar en iOS.

No obstante, tanto xCode para el desarrollo de la aplicación en iOS mediante lenguaje Swift, como diversos entornos de desarrollo para Android, cuentan con la posibilidad de generar emuladores de distintos modelos de smartphones para ambos. Pero se debe tener en cuenta que, por la complejidad del proyecto, así como para las pruebas de funcionamiento real en el dispositivo, sería necesario disponer de un teléfono móvil con conexión bluetooth que se comunicara con los beacons. Hacerlo a través del emulador podría resultar más complejo que directamente usar los medios que tenía a mi alcance; además, tener la posibilidad de aprender a desarrollar aplicaciones en iOS a parte de las vistas hasta ahora durante la carrera en Android. Por estos motivos, la aplicación que usen los usuarios y el administrador del museo estará disponible durante la realización del proyecto para iOS.

2.3.4. SERVIDOR API-REST: NODE.JS

Generalmente, tenemos la concepción que **JavaScript** es un lenguaje de programación del lado del cliente y que se ejecuta en el navegador, y que está relegado a realizar tareas menores. No obstante, actualmente muchos lo catalogan como un lenguaje de programación capaz de realizar lo que cualquier otro lenguaje tradicional como C++, Ruby o Java, con sus peculiaridades y ventajas que lo hacen ideal para determinados usos. Ahora, gracias a **Node.js**, podemos ejecutar JavaScript en el servidor.

Ilustración 14 - Logo de Node.js

Node.js es una **librería y entorno de ejecución** de E/S basado en eventos, por tanto es asíncrona. Aprovechando el motor V8 desarrollado por Google para el uso del navegador Chrome, Node.js es capaz de ejecutar javascript, proporcionando a Node un entorno de ejecución del lado del servidor que compila y ejecutarlo a gran velocidad.

Además de la alta velocidad de ejecución, uno de los aspectos trascendentales de Node.js es lo que se conoce como **Event Loop**, o Bucle de Eventos. Para escalar grandes volúmenes de clientes, todas las operaciones I/O en Node.js se llevan a cabo de forma **asíncrona**. De esta forma, cuando una aplicación Node.js necesita realizar una operación de bloqueo, envía una tarea asíncrona al event loop, junto con un **callback**, y luego continúa. De esta forma, para un proyecto en el que varios usuarios lanzan varias peticiones contra el servidor, es importante cumplir con una eficiencia que otras aplicaciones más generalizadas y basadas en hilos no lo harían.

Desarrollando más el tema del párrafo anterior, Node.js tiene como principal objetivo el proporcionar una manera sencilla para construir programas de red escalables. En lenguajes más "tradicionales" como **Java** o **PHP**, cada conexión genera un nuevo hilo que potencialmente se acompaña de 2MB de memoria. Si tuviéramos un sistema de, por ejemplo, 8 GB de Ram, da un número máximo teórico de conexiones concurrentes de alrededor de 4.000 usuarios. Como veremos más adelante, la cantidad de consultas y peticiones de usuarios de un mismo museo a la vez, puede ser bastante elevada. Por ello, he decidido recurrir a Node, que en lugar de generar un nuevo hilo de Sistema Operativo para cada conexión y asignar su correspondiente memoria, crea con cada conexión un evento dentro del proceso del motor de Node. Además, puesto que no se producen bloqueos gracias al Event Loop, podemos afirmar que nunca se quedará en punto muerto.

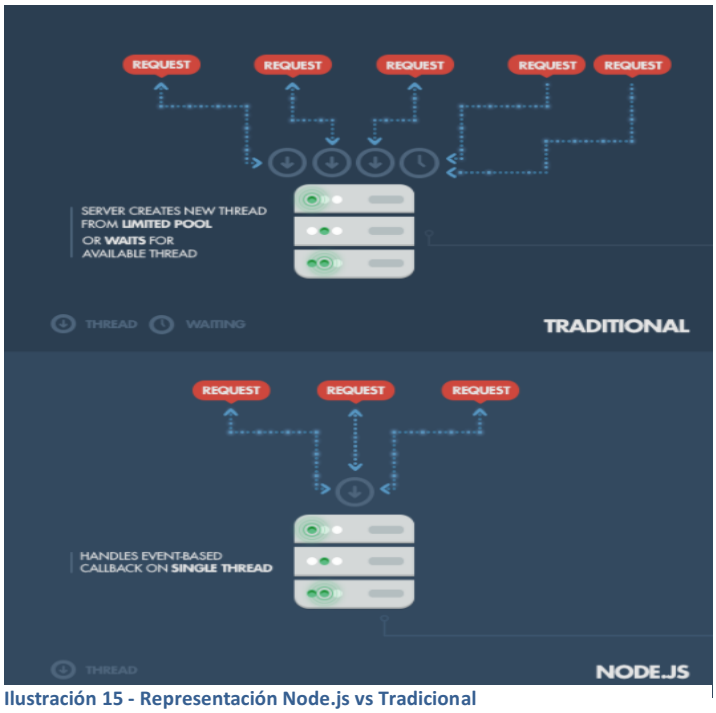


Ilustración 15 - Representación Node.js vs Tradicional

Otro aspecto que debemos tener en cuenta, es el valor de javascript como lenguaje: es conocido por millones de desarrolladores, por lo que la curva de aprendizaje para el uso de Node se vería reducida.

Uno de los valores más importantes con los que cuenta Node, es la amplia comunidad. Esto hace que ante errores comunes de uso podamos afirmar que se encuentra perfectamente depurado o que podamos encontrar soluciones en los problemas con los que nos vayamos topando durante el desarrollo.

Por tanto, dado que nuestra solución esperamos que tenga gran multitud de consultas, y que varios usuarios pueden estar requiriendo del servidor de forma **concurrente**, debemos elegir un entorno que trate de la mejor manera este tipo de conexiones, y que sea escalable. Además, debemos tener en cuenta que los datos que se intercambien no serán paquetes de grandes dimensiones: generalmente será un identificador, una autenticación de usuario, algún update de información, etc. por lo que no necesitamos un modelo que tenga en cuenta un intercambio de datos pesados, si no que sea capaz de hacer muchos a gran velocidad. Se debe tener en cuenta de igual manera, que actualmente esta es una idea muy extendida en el terreno de la informática con nuevos paradigmas como son el Big Data, donde se obtienen gran cantidad de datos de distintas fuentes y que se deben poder **almacenar rápidamente y de forma uniforme**. Por ello, creo que elegir para el desarrollo de mi trabajo de fin de grado este tipo de entorno puede ser muy valioso para el futuro, a parte de la obvia justificación de elección del mismo.

Aunque se trate un poco más adelante, al estar tan relacionado con el servidor, debemos hacer mención al uso de **MongoDB** como base de datos No-SQL, que permite, como hemos explicado anteriormente, almacenar y extraer datos de forma rápida y en un formato determinado. Node.js trabaja perfectamente con este tipo de base de datos al no tener que preocuparme de la conversión entre JSON y cualquier cosa que debamos leer o escribir desde la base de datos. Podemos evitar la necesidad de realizar varias conversiones mediante un formato de serialización de datos uniformes a través de cliente, servidor, y base de datos. Pero lo relacionado con MongoDB, lo trataremos en el siguiente apartado.

No obstante, a parte de la justificación de la elección de Node.js como entorno para el servidor, vamos a puntualizar cuales han sido las alternativas que se han barajado.

En primer lugar, barajé la posibilidad de usar PHP para el código del lado del servidor. PHP es un lenguaje de programación open source cuyo propósito general es el código del lado de servidor, originalmente diseñado para el desarrollo web de contenido dinámico. Debía tenerlo en cuenta, por lo menos, por la gran cantidad se sitios que lo utilizan. Su principal característica es que la ejecución del código se realiza en el servidor, generando un HTML y enviando este al cliente. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas. No obstante, cada conexión que se realice consume una pequeña parte de los recursos del servidor. Esto sería positivo

para una aplicación que no tenga muchas conexiones, ya que cada conexión sería independiente del resto, por lo que, si una de ellas es demasiado lenta, lo sería únicamente para sí misma. El problema reside en el momento en el que hay muchas conexiones, quedando el servidor colapsado y sin recursos. En Node.js ocurre todo lo contrario: tiene un uso mucho más eficiente de los recursos y una respuesta más rápida a conexiones simultáneas pero una tarea pesada repercutiría en lentitud para todo el servidor. Dado el caso de uso de nuestro proyecto, y dado que las tareas no serían en ningún caso pesadas, únicamente de enrutado, modelo de base de datos y consultas a la misma, es preferible conseguir buena conexión simultánea y descartar por tanto PHP.

Otra posibilidad era usar Python en vez de javascript. Una vez más, la escalabilidad fue uno de los principales problemas encontrados. Si bien ahora en Python 3 es posible realizar **I/O asíncronas** gracias a asyncio, pero la mayoría de opciones no se encuentra lo suficientemente madura y en muchos casos es necesario recurrir a librerías de terceros.

No obstante, dando una visión y un razonamiento cuantitativo ante el descarte de Python en favor de Node.js y su Javascript, he consultado diferentes estudios realizados testeando ambos. Concretamente, he localizado un interesante comparativo a través del blog cr0hn, un blog personal escrito por Daniel García, Investigador Senior de Seguridad y Penetration Tester, con artículos de divulgación informática muy técnicos e interesantes sobre diversos temas sobre esta ciencia. En su artículo Rendimiento: Python vs Node.js, realiza diferentes pruebas de carga y rendimiento. Para ello utiliza Node.js con el módulo Express, y Python con **asyncio**. Según lo visto anteriormente, esta sería la configuración de Python más asíncrona y que más podría amoldarse a nuestro modelo de aplicación. Para ello se usa este pequeño código por el cual se devuelve un HTML, buscando medir la cantidad de peticiones que pueden atender por segundo, tiempo total empleado en procesar X peticiones, y cantidad de clientes concurrentes que son capaces de atender.

Python

```python
import asyncio

from aiohttp import web


@asyncio.coroutine
def home(request):
    return web.Response(body=b"""<!DOCTYPE html>
<html lang="es">
<head>
  <title>Hello world!</title>
</head>
<body>
<p>P&aactue;gina de prueba</p>
</body>
</html>""")


@asyncio.coroutine
def json_path(request):
    return web.json_response({"test": 1, "data": "hello world" })


def main():
    app = web.Application()
    app.router.add_route('GET', '/', home)
    app.router.add_route('GET', '/json', json_path)

    web.run_app(app, port=8081, backlog=5000)

if __name__ == '__main__':
    main()
```

NodeJS

```nodejs
const express = require('express');

// Constants
const PORT = 8080;

// App
const app = express();

// End-points
app.get('/', function (req, res) {
  res.send(`<!DOCTYPE html>
<html lang="es">
<head>
  <title>Hello world!</title>
</head>
<body>
<p>Página de prueba</p>
</body>
</html>`);
});


app.get('/json', function (req, res) {
  res.send(JSON.stringify({ test: 1, data: "hello world" }));
});

// Start server
app.listen(PORT, backlog=5000);

console.log('Running on http://localhost:' + PORT);
```

Ilustración 16 - Programas de prueba

Los resultados son los siguientes:

TIEMPO EN PROCESAR TOTAL DE PETICIONES.		
Peticiones	NodeJS	Python
2.000	0.99 seg	2.02 seg
5.000	2.57 seg	5.49 seg
10.000	4.81 seg	12.85 seg

Tabla 13 - Tiempo de proceso de peticiones

PETICIONES POR SEGUNDO		
Peticiones	**NodeJS**	**Python**
2.000	2.072 req/seg	1.005 req/seg
5.000	1.971 req/seg	915 req/seg
10.000	2.084 req/seg	779 req/seg

Tabla 14 - Peticiones por Segundo

TIEMPO REQUERIDO EN ATENDER X CLIENTES (Sin parches de Python y sin propiedad de backlog de NodeJS)		
Peticiones	**NodeJS**	**Python**
100	3.18 seg	9.82 seg
250	3.10 seg	-
500	10.75 seg	-

Tabla 15 - Tiempo requerido en atender X clientes

Como se puede observar, el **rendimiento** de Node.js es muy superior. Hay que tener en cuenta que el tiempo requerido en atender un número determinado de clientes se ha realizado sin parches de Python y sin la propiedad de backlog de Node.js, que viene configurado con 512 conexiones concurrentes por defecto pero que se puede ampliar simplemente inicializando esta variable backlog. Debemos tener en cuenta que aquellos resultados marcados con un guion, significa que el servidor rechaza conexiones y no soporta el flujo de información.

Por tanto, al tener planteada una aplicación que requiera una **cantidad de consultas elevada**, pero de poco nivel computacional y paquetes de **datos pequeños**, y en vista a los resultados, queda justificada la elección de Node.js como entorno para el desarrollo del servidor.

2.3.5. BASE DE DATOS NOSQL: MONGODB

Elegir una base de datos es un asunto que cada vez más influye en el rendimiento de la aplicación. A parte de la cantidad de oferta de base de datos que ha surgido en lo últimos tiempos, está despuntando, aunque no es nuevo, las bases de datos NoSQL como alternativa a las bases de datos SQL como las más extendidas.

Las bases de datos SQL, por sus siglas en inglés **Structured Query Language** es un lenguaje específico que da acceso a un sistema de gestión de bases de datos relacionales. Consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos, y un lenguaje de control de datos. Incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas, y el control de acceso a los datos.

Por su parte, las bases de datos NoSQL (Not only SQL) es una amplia clase de sistemas de gestión de bases de datos que difieren de las bases de datos relacionales en aspectos importantes, siendo el más destacado que no usan SQL en sus consultas. Los datos **no requieren estructuras fijas** como tablas tal y como las comprendemos en las bases de datos relacionales.

Comparando las características de una y otra, justificaré la elección de tipo de base de datos NoSQL como la utilizada para almacenar los datos para el uso de nuestra aplicación.

SQL	NOSQL
Está más adaptado su uso, son mayoritarios y más baratos.	Escalabilidad y carácter descentralizado. Soportan estructuras distribuidas.
Debido al largo tiempo que llevan en el mercado, tienen más soporte, mejores suites de productos y add-ons para gestionar estas bases de datos.	Mayor adaptación a necesidades de proyectos que los modelos de entidad-relación.
Atomicidad de las operaciones en las bases de datos.	Se pueden hacer cambios en los esquemas sin tener que parar la base de datos.
Deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad.	Escalabilidad horizontal: capaces de crecer en número de máquinas en lugar de tener que residir en grandes máquinas.
Ideal para proyectos en los que los datos deben ser consistentes sin dar posibilidad al error.	Se pueden ejecutar en máquinas con pocos recursos y, por tanto, más baratas.
Atomicidad de las operaciones repercuten en el rendimiento de la base de datos.	Optimización de consultas en base de datos para grandes cantidades de datos.

Escalabilidad menor que en las bases de datos NoSQL	Proyectos en los que las estructuras de datos que manejamos son variables.
	Análisis de grandes cantidades de datos en modo lectura.
	Captura y procesado de eventos, con motores de inteligencia complejos.

Tabla 16 - Comparativo SQL y NoSQL

Característica	SQL	NOSQL
Rendimiento	Baja	Alta
Fiabilidad	Buena	Pobre
Disponibilidad	Buena	Buena
Consistencia	Buena	Pobre
Almacenamiento de Datos	De tamaño medio a grande	Optimizado para gran cantidad de datos
Escalabilidad	Alta, pero cara	Alta

Tabla 17 - Comparativo SQL y NoSQL (II)

Por tanto, dado que la inteligencia de consulta y tratamiento de datos en la base de datos no residirá en la propia base de datos, y que se limitará a almacenar los datos y hacer efectivas las consultas GET, POST, PUT y DELETE, debemos centrarnos en el valor que nos aporta una **mayor escalabilidad** y los rápidos cambios. En nuestra aplicación realizaremos una gran cantidad de consultas, de tipo GET y PUT, pudiendo ser una gran cantidad por segundo, y de datos de pequeño tamaño, sin necesidad de ser estructurados más allá de un formato de datos JSON. Además de ser más baratas que las bases de datos relacionales, son bases de datos que están muy orientados a guardar cualquier tipo de dato, sea cual sea su formato, incluso binario, sin necesidad que sea igual a un formato predefinido como se necesitaría en una base de datos estructurada. Este hecho puede aportar valor a este proyecto y dar la oportunidad de aprender a usar este tipo de bases de datos.

Una vez decidido que la base de datos a utilizar será de tipo NoSQL, vamos a analizar la elección de cuál seleccionar entre la creciente oferta de este tipo de bases de datos.

Además de ser una base de datos NoSQL, tenemos varios subtipos dentro de este tipo de bases de datos, dependiendo de la forma en la que se almacena la información. De esta forma, tenemos los siguientes tipos de base de datos NoSQL

BASES DE DATOS CLAVE-VALOR

Son el modelo de base de datos NoSQL más popular, además de ser la más sencilla en cuanto a funcionalidad. En este modelo tenemos contenedores, también conocido como cabinets, donde podemos tener tantas parejas de clave-valor como queramos. En cada contenedor podemos tener datos de una misma naturaleza (productos, clientes, pedidos, etc.) o totalmente diferente (por ejemplo, un contenedor por cliente). Cada elemento está identificado por una **llave única**, lo que permite recuperar la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario BLOB). Muy eficientes tanto para lecturas como escrituras.

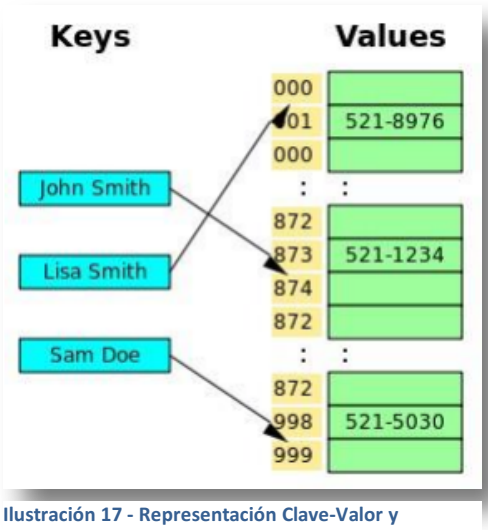


Ilustración 17 - Representación Clave-Valor y Ejemplos

cassandra

Google BigTable

APACHE HBASE

ORACLE NOSQL DATABASE

BASES DE DATOS DOCUMENTALES

Este tipo de base de datos NoSQL almacena la información como un **documento**, generalmente usando una estructura para los mismos en formato **JSON** o **XML**, y donde se puede utilizar una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave-valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles.

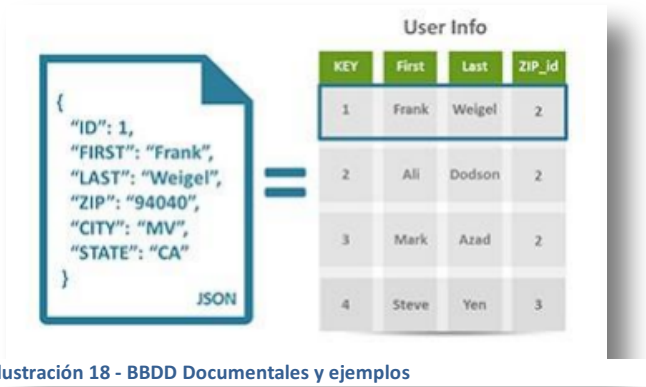


Ilustración 18 - BBDD Documentales y ejemplos

mongoDB

{ name: mongo, type: DB }

CouchDB relax

BASES DE DATOS EN GRAFO

En este tipo de base de datos, la información se representa como nodos de un **grafo**, y sus relaciones, con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. La complejidad de este tipo de base de datos reside en su estructura, que debe estar totalmente normalizada, de tal forma que cada tabla tenga una sola columna y cada relación, dos.

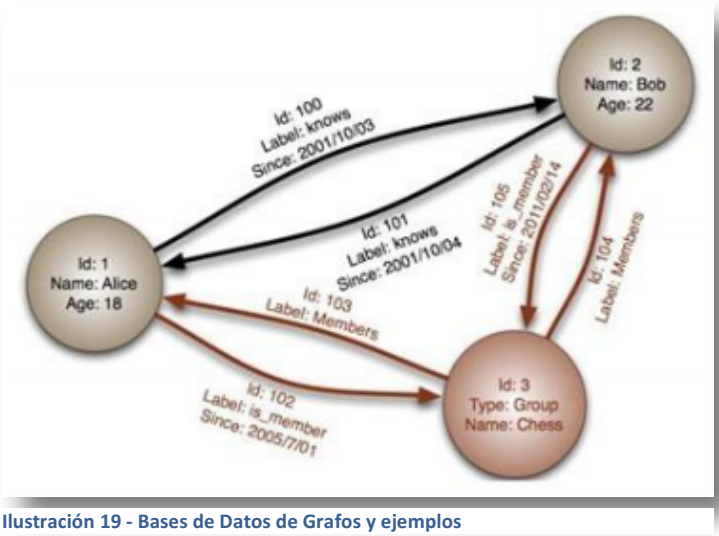


Ilustración 19 - Bases de Datos de Grafos y ejemplos

BASE DE DATOS ORIENTADAS A OBJETOS

En este tipo, la información se representa mediante **objetos** de la misma forma que podría hacerse en lenguajes de programación orientados a objetos como JAVA, C# o .NET.

Ilustración 20 - Bases de Datos Orientadas a Objetos y Ejemplos

CONCLUSIÓN

Dada la naturaleza de nuestro proyecto, y teniendo en cuenta la curva de aprendizaje para cada uno de los modelos vistos de bases de datos NoSQL, podemos descartar las bases de datos por grafos, y las bases de datos orientadas a objetos. Este último caso es, por el resto de elementos elegidos para el desarrollo de la aplicación, encaja otro modelo de datos que el modelo orientado a objetos. En cuanto a

la elección de clave-valor o documental, considero que los beneficios que buscamos para el correcto funcionamiento de la aplicación, incluso en el caso de tener un número de logs contra el servidor muy elevado, con este tipo de bases de datos NoSQL nos la garantizan ambas.

No obstante, desde la subjetividad, considero que el **modelo documental** se adapta mejor a nuestras necesidades y es más visual y más sencillo a la hora de la implementación: bastaría con aunar todos los datos en un formato JSON y enviarlos a la base de datos. Desde un punto de vista objetivo, intentando dar a nuestra solución las máximas competencias presentes y futuras, si quisiéramos incluir contenido multimedia, audios, etc. la posibilidad de datos binarios como ofrece MongoDB puede ayudarnos a decidirnos por este tipo de base de datos. Además, Node.js, elegida como entorno para desarrollo del APIRest del servidor, trabaja perfectamente con esta base de datos, y es la que mayor crecimiento ha tenido en los últimos años, tanto, que en vista al crecimiento de este tipo de base de datos en general y de MongoDB en particular, Oracle ha intensificado sus esfuerzos en su base de datos NoSQL, como complemento perfecto para su base de datos relacional y con total integración con toda su pila tecnológica y aportar valor a su solución para el paradigma del Big Data. Es una buena oportunidad para aprender de una tecnología que en el futuro puede ser de gran utilidad.

3. PRODUCTOS Y SOLUCIÓN

3.1. PRIMERA TOMA DE CONTACTO CON LOS BEACONS DE ESTIMOTE.

Uno de los elementos que conforman la solución propuesta en este proyecto, y puede ser que el más importante, son los **beacons**. Los considero como parte fundamentar porque es a través de ellos como vamos a facultar de "conectividad a internet" a determinados objetos, y, por tanto, hacer posible el Internet de Las Cosas. Si bien no tienen conectividad directamente, lo tienen mediante la aplicación móvil con la que se conectan. En cualquier caso, entra dentro del paradigma del IoT.

Para ello, lo primero ha sido adquirir dos kits de desarrollo de Estimote, que contienen 3 beacons cada uno desde la página web www.estimote.com, tardando aproximadamente una semana en llegar al domicilio.

Ilustración 21 - Beacons adquiridos

3.1.1. CONFIGURACIÓN Y PUESTA EN FUNCIONAMIENTO

Una vez abierto el kit adquirido, no es necesario realizar ningún tipo de configuración con respecto al beacon. Lo envían activados, y con las instrucciones para llevar a cabo las pruebas de funcionamiento y familiarizarnos con los dispositivos.

Para ello, disponen de una página web de configuración, donde se pueden observar los beacons adquiridos y que, previo registro durante la compra, quedan vinculados a cada usuario, además de una aplicación móvil para testear como localiza cada uno de los beacons.

En mi caso, la página web recoge la siguiente información:

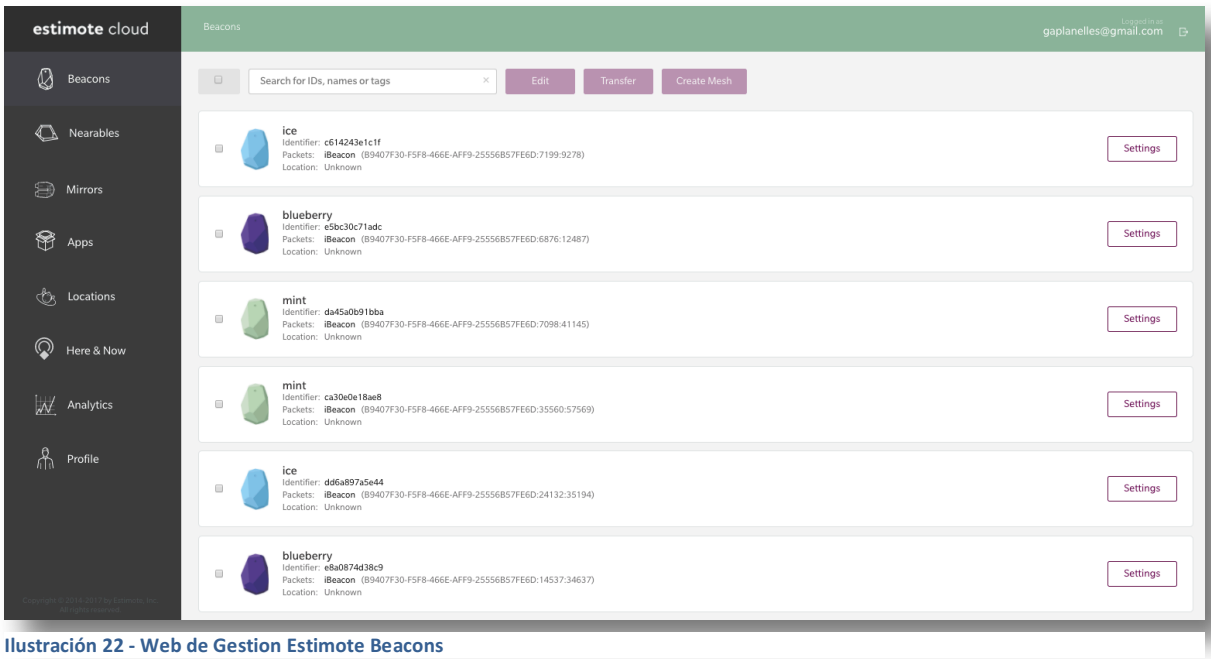


Ilustración 22 - Web de Gestion Estimote Beacons

En dicha pantalla, podemos ver en un menú lateral en el que consultar nuestro perfil, los diferentes dispositivos de los que dispone Estimote y que hemos explicado en el apartado 2.3.2. En nuestro caso, solo hemos comprado beacons, por lo que, pinchando en dicha descripción, podemos ver un resumen de los que tenemos en nuestro poder.

Como podemos observar, se detalla un nombre standard que se atribuye de fábrica en función del color. Todos ellos además disponen de un identificador único, así como un número de serie que finaliza con el **Major Number** del dispositivo, seguido de dos puntos y su **Minor Number**.

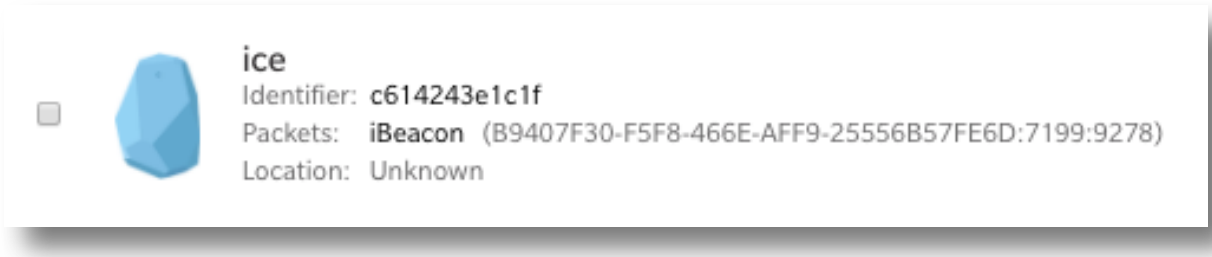


Ilustración 23 - Ejemplo de descripción de Beacons en la Web de Gestión de Estimote

Por ejemplo, en el beacon mostrado anteriormente, su Major Number será el 7199, y el Minor Number el 9278. En cuanto a la localización, en el momento en que lo tengamos emplazado en un lugar geográfico determinado, podemos registrar su paradero para así tener una información detallada de donde tenemos colocado cada uno. En nuestro caso, por ahora en el momento de desarrollo de la solución, no tiene mucho sentido. Además, podemos hacer click en el botón de "Settings" y cambiar todos estos ajustes, a excepción de aquellos que deben ser unívocos, que son los siguientes:

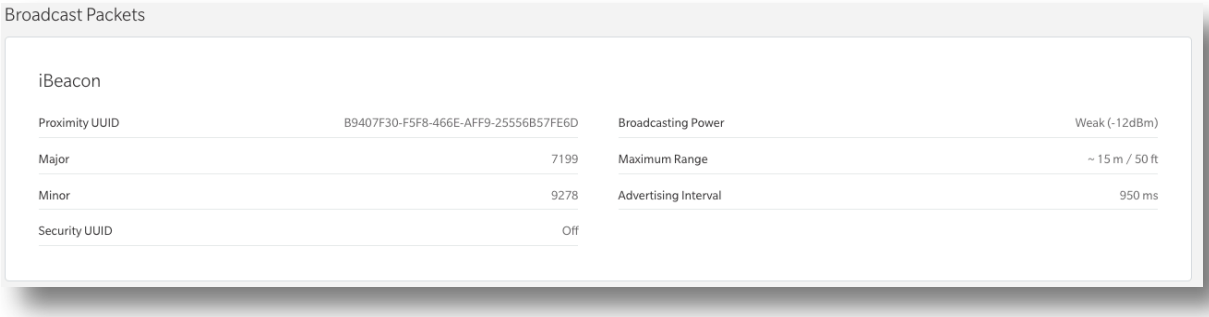


Ilustración 24 - Identificadores Estimote Beacons

Una vez identificado cada uno de nuestro beacons, vamos a probar su funcionamiento con la aplicación móvil para pruebas que poner Estimote al alcance del desarrollador. Para ello, la descargamos del AppStore y la ejecutamos.

Ilustración 25 - Prueba de Estimote Beacons con App de Estimote

Realizando una prueba rápida, podemos ver el funcionamiento de los beacons con la aplicación, identificando en un radar su posición y, al clickar sobre cada uno de ellos, obteniendo la información relevante de identificación, que posteriormente podremos utilizar para realizar una acción vinculada a ese beacon según nuestra inteligencia de negocio.

3.1.2. ¡HOLA, MUNDO! CON BEACONS DE ESTIMOTE: CLOSEST-BEACON-DEMO

Una vez comprobado el modo en que funcionan e interactúan los beacons con la aplicación móvil desde un punto de vista teórico, y con una aplicación en la que no podemos ver el código, lo ideal es crear una aplicación sencilla que ayude a comprender el modo de funcionar.

Para ello, pensé en una aplicación que realizara la conectividad entre los dispositivos bluetooth y la aplicación móvil, y la demostrara, de tal forma que, la parte que tendría la inteligencia de la aplicación la solución de digitalización de museos, fuera sustituida por otra labor más sencilla. Por ejemplo, un "¡Hola, Mundo!" adecuado para una primera aplicación en la que probar el funcionamiento y que sirva de demo para la conectividad app-beacon, sería una aplicación en la que mostrara el **background** del color del beacon que tiene más cerca.

```
//
//  ViewController.swift
//  closest-beacon-demo
//
//  Created by Gonzalo Arévalo Planelles on 26/9/16.
//  Copyright © 2016 Gonzalo Arévalo Planelles. All rights reserved.
//

import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var beaconActual = 0
    let locationManager = CLLocationManager()

    //Región de Beacons en función de los Beacon's UUID que tenemos
    let region = CLBeaconRegion(proximityUUID: UUID(uuidString: "B9407F30-F5F8-466E-
AFF9-25556B57FE6D")!, identifier: "Estimotes")

    //Crear Array de colores en función de Minor Number de 3 de los beacons que
tenemos
    let colors = [
        12487: UIColor(red: 84/255, green: 77/255, blue: 160/255, alpha: 1),
        9278: UIColor(red: 142/255, green: 212/255, blue: 220/255, alpha: 1),
        57569: UIColor(red: 162/255, green: 213/255, blue: 181/255, alpha: 1)
    ]

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        locationManager.delegate = self
        if (CLLocationManager.authorizationStatus() !=
CLAuthorizationStatus.authorizedWhenInUse) {
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.startRangingBeacons(in: region)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
```

```
    func locationManager(_ manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], in region: CLBeaconRegion) {

        //Cogemos el Beacon que esté más próximo
        let knownBeacons = beacons.filter{ $0.proximity != CLProximity.unknown }

        if (knownBeacons.count > 0) {

            //imprimos el beacon más cercano
            let closestBeacon = knownBeacons[0] as CLBeacon
            print(closestBeacon)


            //Mirar por distancia o potencia, cogemos minor number
            let nuevoBeacon = closestBeacon.minor.intValue

            //Si cambia de beacon...
            if beaconActual != nuevoBeacon {
                beaconActual = nuevoBeacon

                //Background del color que corresponda al minor number del beacon más
cercano
                self.view.backgroundColor = self.colors[nuevoBeacon]
                            }
        }
    }
}
```

Ilustración 26 - Código de primera app con beacons

En el código anterior, resultado de la implementación descrita anteriormente, importamos la librería CoreLocation, destinada a servicios de posicionamiento relativo de beacons cercanos. Posteriormente, creamos una región a partir de los UUID de los beacons que tenemos en nuestro poder. Posteriormente, creamos un array de colores, en el que cada color queda identificado por el Minor Number de 3 de los beacons que tenemos. Posteriormente, escaneamos cuál de los beacons es el más cercano, imprimiendo en la consola los detalles del mismo. Para nuestra aplicación, lo que haremos será coger el Minor Number del beacon más cercano y cambiar al background del color que quede identificado por ese mismo número en el array de colores.

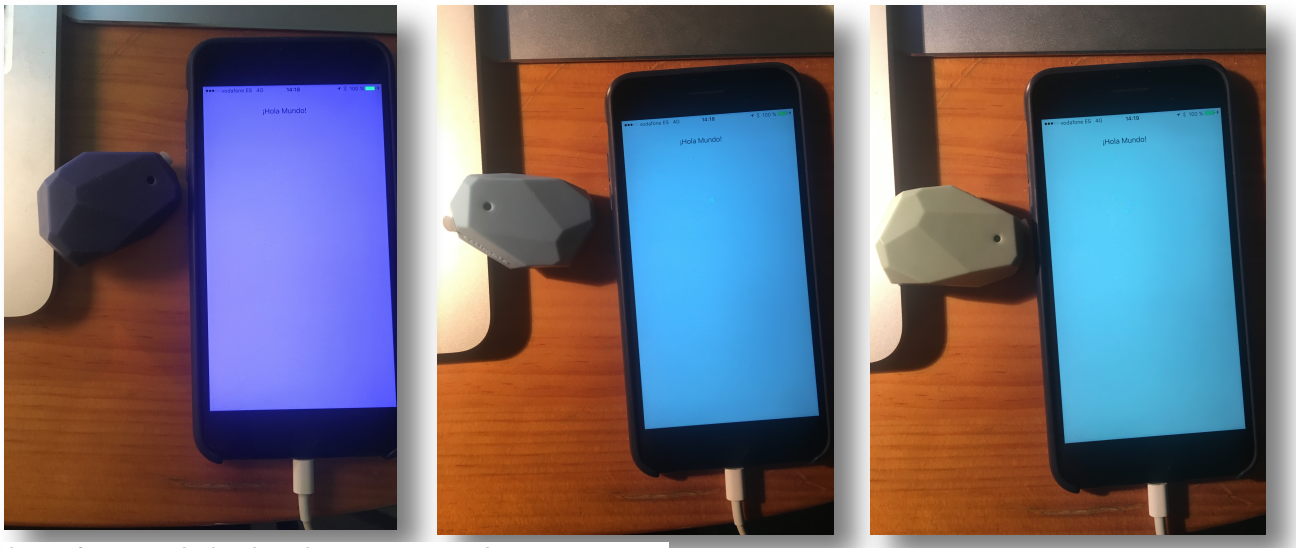


Ilustración 27 - Resultados obtenidos primera app con beacons

Como podemos observar en la aplicación ejecutada en el dispositivo iPhone 7 Plus, **el fondo de la aplicación va cambiando** según tenga más cercano un beacon u otro, vinculado el color que aparece en el background con el color de la cubierta de silicona del beacon, tal y como hemos declarado en el array de colores.

Por último, para terminar con las pruebas de puesta en marcha de los beacons, muestro a continuación las salidas impresas por consola, como teníamos implementado en nuestro código, y demostrar así el funcionamiento:

```
CLBeacon (uuid:B9407F30-F5F8-466E-AFF9-25556B57FE6D, major:35560, minor:57569,
proximity:2 +/- 0.49m, rssi:-70)

CLBeacon (uuid:B9407F30-F5F8-466E-AFF9-25556B57FE6D, major:7199, minor:9278,
proximity:1 +/- 0.77m, rssi:-72)

CLBeacon (uuid:B9407F30-F5F8-466E-AFF9-25556B57FE6D, major:6876, minor:12487,
proximity:1 +/- 0.04m, rssi:-48)
```

Ilustración 28 - Traza impresa por la consola de XCode

Estas salidas corresponden a la impresión del beacon más cercano en cada caso. Como podemos observar, el Minor Number, el número que hemos seleccionado como referencia para identificar el beacon, coincide con los identificadores del array de colores. De esta forma, cambia el fondo de pantalla en función del beacon más cercano.

Si ahora extrapolamos esta implementación a la solución global, donde la aplicación, en lugar de mostrar el fondo de pantalla de un color determinado, realizara una petición a nuestro servidor para solicitar los datos del cuadro vinculado a este beacon, tendríamos probada la conexión entre aplicación móvil y dispositivo bluetooth de manera suficiente como para comenzar a implementar esta otra parte de la solución desde la base implementada como prueba.

3.2. ENTORNO DE DESARROLLO APP MÓVIL IOS: XCODE

Lo descrito en el apartado anterior, no sería posible sin un entorno de desarrollo en el que se pueda implementar, en lenguaje Swift, la aplicación para móvil con sistema operativo iOS. El IDE (Entorno de Desarrollo Integrado, de sus siglas en inglés, Integrated Development Environment) para sistema operativo MacOS, es Xcode.

Para nuestra aplicación móvil, hemos usado lenguaje Swift. Pero no es solo eso. Incluye compiladores para C, C++, Objetive-C, ObjetiveC++, Java y AppleScript. Es digno de señalar su **Interface Builder**, que permite la creación de interfaces de usuario y vincularla con nuestro código de una manera tan fácil como es un "**drag & drop**", de tal forma que es posible realizar un diseño de la interfaz para posteriormente implementar la funcionalidad de cada parte, de una manera muy intuitiva. Esto es una herencia de **NeXT**, empresa fusionada con Apple en 1996.

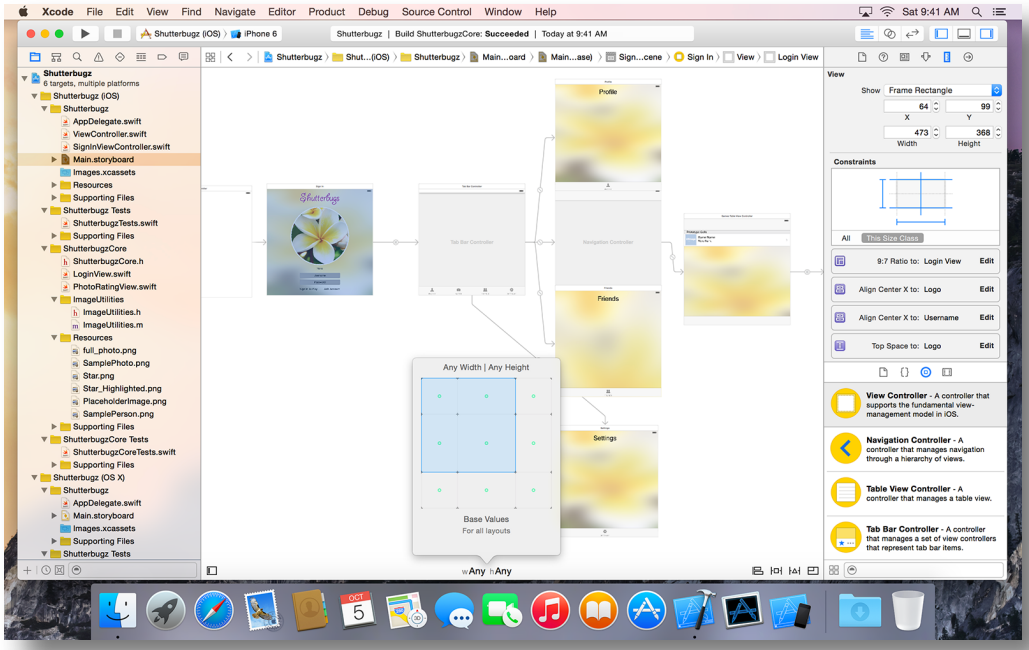


Ilustración 29 - Ejemplo de Storyboard de XCode

Para la instalación y puesta en funcionamiento de este entorno, únicamente es necesario descargar XCode del AppStore.

3.3. ENTORNO DE DESARROLLO DEL SERVIDOR: NODE.JS, NPM, MONGOOSE, EXPRESS Y OTROS

Tal y como se describió en el apartado 2.3.4, el entorno de desarrollo elegido para la implementación del servidor API-Rest ha sido Node.js, usando como lenguaje Javascript.

Antes de comenzar con la escritura de código, ha sido necesario instalar Node.js junto con otros elementos, así como su correcta configuración para el desarrollo del API. A continuación, vamos a detallar los pasos previos que han debido llevarse a cabo para preparar el entorno e implementar la arquitectura que se enuncia en apartados siguientes de esta memoria.

3.3.1. INSTALACIÓN DE NODE.JS Y GESTOR DE PAQUETES NPM

Como hemos explicado anteriormente, Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Para su instalación debemos dirigirnos a su página web www.nodejs.org y descargar e instalar la versión de largo soporte (Long Term Support: **LTS**). Junto con ello, instalamos **npm**, que es el gestor de paquetes de Node.js. Esto es muy importante pues permite instalar librerías de terceros, así como las propias del **core** de Node. Para este paso, podemos usar el instalador que se descarga desde la web.

Una vez instalado, compruebo la versión instalada, como forma de verificar que es la correcta.

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ node -v
v6.9.4
MacBook-Pro-de-Gonzalo:api-rest Gonza$
```

Ilustración 30 - Versión de Node.js

Al día de comienzo del proyecto y del desarrollo de la API, la versión 6.9.4 era la versión LTS de Node.js. Puede haber cambiado, pero no en el número mayor, como mucho en pequeños cambios o corrección de errores. De la misma forma, verifico la versión de npm instalada:

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ npm -v
3.10.10
MacBook-Pro-de-Gonzalo:api-rest Gonza$
```

Ilustración 31 - Versión de npm

Una vez finalizada la instalación y comprobada la versión de la misma, es conveniente realizar una pequeña prueba para testar el correcto funcionamiento de Node.js. Para ello, abrimos el **intérprete** de Node y ejecutamos un pequeño código javascript.

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ node
> 3+3
6
> console.log('Hola, Mundo')
Hola, Mundo
undefined
> var a = 5
undefined
> a
5
>
```

Ilustración 32 - Pruebas de compilación y ejecución de código Javascript

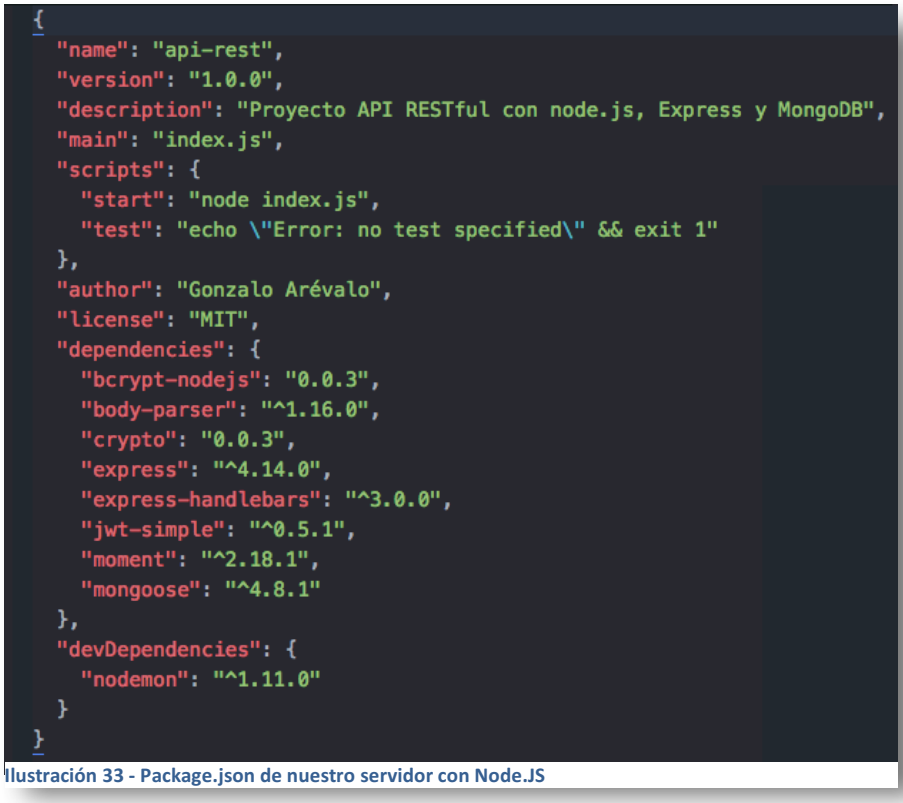
Como podemos observar, una vez ejecutamos Node.js con el comando node, se abre el intérprete de comandos donde, incluyendo código javascript, lo **interpreta y** lo **ejecuta** correctamente. Esto nos sirve para saber que ante una serie de documentos javascript pasados adecuadamente a Node, tendrá una correcta interpretación y ejecución.

3.3.2. INSTALACIÓN DE EXPRESS Y ATOM COMO EDITOR DE TEXTO

Antes de comenzar con la implementación del código back-end del servidor, debemos añadir una serie de componentes que garanticen un correcto funcionamiento de Node.js. En primer lugar, por comodidad y por haberlo utilizado anteriormente para programación web, como editor de texto he elegido **Atom**. No difiere mucho de un editor de texto tradicional, capaz de editar texto con extensión js, html, css, etc.

Se desarrollará el API Rest con un pequeño fichero js que hará sus veces de "servidor interno", en el sentido que se consultará a este para comprobar el correcto funcionamiento de node.js y el resto de componentes, previamente a añadir funcionalidades más propias de la solución propuesta.

Para ello, se creará en Atom la carpeta en el directorio que deseemos. Posteriormente, en la terminal y en el mismo directorio, ejecutaremos en la terminal "npm init". Esto solicitará una serie de información al programador relativa al proyecto, que formará un archivo **package.json** que resumirá los componentes del proyecto y sus aspectos más relevantes. Este archivo, tiene la siguiente forma:



```
{
  "name": "api-rest",
  "version": "1.0.0",
  "description": "Proyecto API RESTful con node.js, Express y MongoDB",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Gonzalo Arévalo",
  "license": "MIT",
  "dependencies": {
    "bcrypt-nodejs": "0.0.3",
    "body-parser": "^1.16.0",
    "crypto": "0.0.3",
    "express": "^4.14.0",
    "express-handlebars": "^3.0.0",
    "jwt-simple": "^0.5.1",
    "moment": "^2.18.1",
    "mongoose": "^4.8.1"
  },
  "devDependencies": {
    "nodemon": "^1.11.0"
  }
}
```

Ilustración 33 - Package.json de nuestro servidor con Node.JS

En packaje.json podemos ver el nombre del proyecto, su versión, una breve descripción, cual es el archivo al que debe apuntar para comenzar con la ejecución (en este caso index.js) y con qué script debe ejecutarse (llamando a node pasándole index.js, como se puede ver en el atributo Scripts, en el subatributo "start").

En este archivo es importante mencionar las dependencias, donde se detallan pequeñas librerías y frameworks que ayudarán a realizar algún trabajo en concreto. "**Crypto**" y "**bcrypt**" son utilizadas, por ejemplo, para la encriptación de claves. El resto, son librerías y **frameworks** en los que debemos detenernos un momento para explicar el porqué de su uso.

EXPRESS

Para implementar el API Rest que hemos ideado, necesitamos una comunicación cliente-servidor (app móvil-servidor API Rest) con verbos **HTTP**. Express es un framework de Node.js que nos facilita este tipo de comunicación.

Para instalarlo, puesto que es un framework de Node, podemos usar **npm**. Con la directiva "npm install express --save" tendremos instalado Express y aparecerá en las dependencias de nuestro package.json.

BODY-PARSER

Esta librería, en un principio, estaba incluida dentro de **Express**. Posteriormente fue separada por lo que es necesario instalarla por separado. No tiene más dificultad que usar de nuevo npm para su instalación, con la directiva "npm install body-parser --save".

En el momento en que enviemos al servidor peticiones POST o PUT, que son típicamente las que reciben parámetros (aunque sirve para cualquiera de las peticións HTTP Rest), nos permite **parsear el cuerpo de la petición** y leer esos datos que enviemos, de tal forma que Node.js pueda tratarlos y ser guardados en una base de datos.

NODEMON

Como pasa con Body-Parser, es una librería que se debe instalar con el gestor de paquetes npm, ya que en la actualidad no se añade a Express como pasaba con anterioridad. Esta es una herramienta útil para el desarrollador, pues detecta el momento en que se guarda un archivo para reiniciar el servidor, y poder ver si contiene un error o no. No es de vital importancia para la correcta codificación del servidor, pero

ha sido una herramienta de gran utilidad y que me ha ahorrado tiempo y ha ayudado en la depuración de errores.

MONGOOSE

Este es uno de los elementos más importantes para el correcto funcionamiento de toda la comunicación entre el servidor y la base de datos. Mongoose es el equivalente a un ORM (Object-Relational Mapping de sus siglas en inglés) en las bases de datos relacionales habituales, es decir, un mapeo objeto relacional, consistente en convertir datos entre el sistema de tipos utilizado en el lenguaje de programación para adecuarlo al de la base de datos, en nuestro caso, MongoDB.

Node.js **dispone de un driver nativo** para MongoGB, pero con esta librería estamos una capa por encima de este driver y nos permite crear esquemas, conectar con la base de datos de una manera más sencillas, etc.

Para instalarla usamos el gestor de archivos: "npm i –S mongoose".

Mongoose dispone de un método **connect** que, al enviarle la URL de donde se encuentra la base de datos en un String, y, con su correspondiente callback, nos permite conectarnos a la base de datos y empezar a ejecutar toda la aplicación.

3.3.3. INSTALACIÓN DE MONGODB

Para la instalación de MongoDB en Mac vamos a utilizar "**homebrew**", que es un gestor de paquetes similar al que usa Ubuntu con su apt-get install o npm install en el caso de Node.js. Homebrew funciona de una manera similar para Mac. Para el caso concreto de MongoDB, nos permite instalarlo de una forma rápida y fácil.

Para ello, primeramente, instalamos Homebrew copiando la URL disponible en su web https://brew.sh en nuestra terminal. Una vez instalado, ejecutamos "brew install mongodb".

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ brew upgrade mongodb
==> Upgrading 1 outdated package, with result:
mongodb 3.4.9
==> Upgrading mongodb
==> Installing dependencies for mongodb: openssl
==> Installing mongodb dependency: openssl
==> Downloading https://homebrew.bintray.com/bottles/openssl-1.0.2l.sierra.bottl
==> Downloading from https://akamai.bintray.com/b9/b9a6d41e2889890de8db396c2c280
######################################################################## 100,0%
==> Pouring openssl-1.0.2l.sierra.bottle.tar.gz
==> Caveats
A CA file has been bootstrapped using certificates from the SystemRoots
keychain. To add additional certificates (e.g. the certificates added in
the System keychain), place .pem files in
  /usr/local/etc/openssl/certs

and run
  /usr/local/opt/openssl/bin/c_rehash

This formula is keg-only, which means it was not symlinked into /usr/local,
because Apple has deprecated use of OpenSSL in favor of its own TLS and crypto l
ibraries.

If you need to have this software first in your PATH run:
  echo 'export PATH="/usr/local/opt/openssl/bin:$PATH"' >> ~/.bash_profile

For compilers to find this software you may need to set:
    LDFLAGS:  -L/usr/local/opt/openssl/lib
    CPPFLAGS: -I/usr/local/opt/openssl/include

==> Summary
🍺  /usr/local/Cellar/openssl/1.0.2l: 1,709 files, 12.2MB
==> Installing mongodb
==> Downloading https://homebrew.bintray.com/bottles/mongodb-3.4.9.sierra.bottle
==> Downloading from https://akamai.bintray.com/ab/abba5a957f15d3a1d46762e9834ca
######################################################################## 100,0%
==> Pouring mongodb-3.4.9.sierra.bottle.tar.gz
==> Caveats
To have launchd start mongodb now and restart at login:
  brew services start mongodb
Or, if you don't want/need a background service you can just run:
  mongod --config /usr/local/etc/mongod.conf
==> Summary
🍺  /usr/local/Cellar/mongodb/3.4.9: 19 files, 284.9MB
MacBook-Pro-de-Gonzalo:api-rest Gonza$
```

Ilustración 34 - Instalación de MongoDB

Una vez instalado, para comprobar su funcionamiento, entendiendo que el desarrollo primeramente se hará en local, y posteriormente se alojará en un servidor cloud, debemos tener una terminal corriendo MongoDB para poder interactuar con ella mediante una segunda terminal.

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ sudo mongod
Password:
2017-09-18T18:06:15.970+0200 I CONTROL  [initandlisten] MongoDB starting : pid=6
4834 port=27017 dbpath=/data/db 64-bit host=MacBook-Pro-de-Gonzalo.local
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] db version v3.4.9
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] git version: 876ebee8c7d
d0e2d992f36a848ff4dc50ee6603e
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2l  25 May 2017
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] allocator: system
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] modules: none
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] build environment:
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten]     distarch: x86_64
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten]     target_arch: x86_64
2017-09-18T18:06:15.971+0200 I CONTROL  [initandlisten] options: {}
2017-09-18T18:06:15.972+0200 W -        [initandlisten] Detected unclean shutdow
n - /data/db/mongod.lock is not empty.
2017-09-18T18:06:15.972+0200 I -        [initandlisten] Detected data files in /
data/db created by the 'wiredTiger' storage engine, so setting the active storag
e engine to 'wiredTiger'.
2017-09-18T18:06:15.972+0200 W STORAGE  [initandlisten] Recovering data from the
 last clean checkpoint.
2017-09-18T18:06:15.973+0200 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=3584M,session_max=20000,eviction=(threads_min=4,threads_max=4)
,config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal
,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,lo
g_size=2GB),statistics_log=(wait=0),
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits
 too low. Number of files is 256, should be at least 1000
2017-09-18T18:06:16.746+0200 I FTDC     [initandlisten] Initializing full-time d
iagnostic data capture with directory '/data/db/diagnostic.data'
2017-09-18T18:06:16.747+0200 I NETWORK  [thread1] waiting for connections on por
t 27017
2017-09-18T18:06:17.011+0200 I FTDC     [ftdc] Unclean full-time diagnostic data
 capture shutdown detected, found interim file, some metrics may have been lost.
 OK
```

```
MacBook-Pro-de-Gonzalo:api-rest Gonza$ mongo
MongoDB shell version v3.4.9
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.9
Server has startup warnings:
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten]
2017-09-18T18:06:16.689+0200 I CONTROL  [initandlisten] ** WARNING: soft rlimits
 too low. Number of files is 256, should be at least 1000
> showdbs
2017-09-18T18:08:51.063+0200 E QUERY    [thread1] ReferenceError: showdbs is not
 defined :
@(shell):1:1
> show dbs
local   0.000GB
museum  0.000GB
test    0.000GB
>
```

Ilustración 35 - Prueba de Funcionamiento de MongoDB

En el cuadro anterior podemos observar un ejemplo de funcionamiento de la base de datos NoSQL MongoDB que, mientras que está corriendo en la terminal de la izquierda, podemos conectarnos a la misma y solicitar que nos muestre las bases de datos con las que cuenta en local. Como podemos observar, existe la base de datos "museum", que ha sido sobre la que se han realizado pruebas para este trabajo.

3.3.4. POSTMAN

En este subapartado únicamente se mencionará el uso de Postman. Es un entorno de desarrollo de API's que tiene la capacidad de actuar como cliente en una comunicación con un servidor, de tal forma que se puede enviar cualquier tipo de verbo HTTP, recibir la respuesta del servidor, enviar parámetros, atributos en el cuerpo del mensaje… Gracias a esta herramienta se ha podido desarrollar el servidor API Rest y realizar pruebas que pudieran finalizarlo y desplegarlo en un cloud sin necesidad de desarrollar un cliente alternativo (o la aplicación cliente final) asegurándonos de su correcto funcionamiento antes de esto.

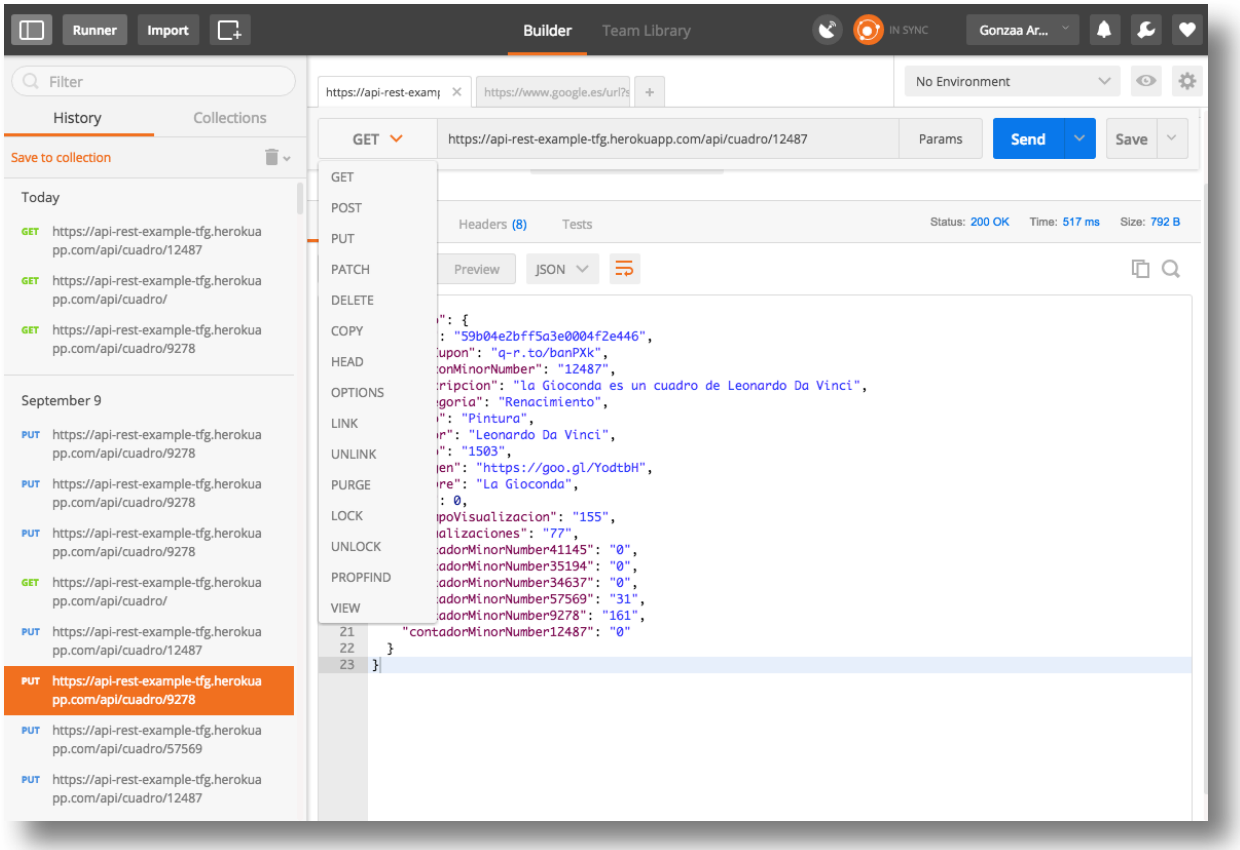


Ilustración 36 - Ejemplo de funcionamiento de Postman

3.4. ANÁLISIS DEL SISTEMA

En este apartado se obtendrán las especificaciones del sistema software a desarrollar. El objetivo es, sin entrar en detalles de diseño, recopilar todas aquellas funcionalidades que debe tener la solución. Se debe procurar seguir una minuciosidad y rigurosidad para evitar dejar funcionalidades importantes fuera o incluir especificaciones que sean incompatibles unas con otras.

3.4.1. DEFINICIÓN DEL SISTEMA

En primer lugar, antes de la recopilación de requisitos del sistema, habrá que concretar la definición de este. Se deberá delimitar su alcance, es decir, el problema que resuelve el sistema; además las restricciones a las que estará sometido y el entorno operacional, esto es, en qué dispositivos será posible su ejecución.

ALCANCE

El sistema a desarrollar consiste en una solución capaz de **digitalizar** un museo o espacio de exposiciones, de tal manera que, a través de una **aplicación móvil** que detecta qué obras expuestas tiene más cerca a partir de un dispositivo beacon vinculado a cada una de ellas, se otorgue por un lado al visitante de la capacidad de **interactuar** con el museo, y al museo gestionar estas vinculaciones beacon-obras expuestas, el flujo de visitantes en el interior del museo, y gestionar descuentos y métodos de **maximización de ventas** de la tienda de souvenirs.

RESTRICCIONES

A continuación, se muestran un conjunto de restricciones impuestas al proyecto ya sea por el planteamiento general o por los medios a disposición para el desarrollo del mismo:

- El **software** que controla la aplicación móvil, tanto de usuario como de administración del museo, y la conexión con los dispositivos beacons estará programado en lenguaje Swift.
- El servidor, que será tipo **API-Rest**, estará programado en lenguaje javascript.
- Toda la comunicación de ambas aplicaciones móviles se realizará mediante verbos http, y los datos serán enviados en formato JSON, para facilitar su parseado y su almacenamiento en base de datos **NoSQL** de tipo **documental**.
- La aplicación de usuario debe poder ejecutarse en el mayor número de terminales iOS posible.

- Las contraseñas de usuario y administrador serán cifradas mediante **algoritmo criptográfico** seguro. La comparación de contraseña almacenada e introducida se realizará mediante comparación criptográfica, nunca decodificando la clave almacenada.
- Todas las funcionalidades extraídas de los requisitos de usuario deben implementarse en el sistema.

ENTORNO OPERACIONAL

En el presente apartado, se especificará en qué dispositivos se ejecutará cada una de las partes del sistema.

- La dotación de internet a los distintos objetos del museo se realizará mediante dispositivo Beacon de marca Estimote, por sus posibilidades de modularidad, amplia oferta de artículos en función de distintas necesidades, y capacidad de **ampliar funcionalidades** que podrían ser de interés en el futuro.
- La aplicación móvil se ejecutará en cualquier dispositivo iOS, mientras que esté actualizado a última versión iOS 10 o compatible, tanto en **smartphones** como **tablets**.

3.4.2. REQUISITOS DE USUARIO

Los requisitos de usuario provienen de decisiones propias o de razonamientos llevado a cabo junto con los tutores en las tutorías llevadas a cabo. Si bien estos requisitos deberían ser estipulados por el cliente del proyecto, al ser un trabajo de fin de grado, me veo obligado a tomar estas decisiones de manera propia.

Los requisitos los dividiremos en requisitos de capacidad y requisitos de restricción. A continuación, explicamos los diversos apartados de los que constan cada uno de ellos:

- **Identificador:** Identificador unívoco del requisito. Cada uno constará de una nomenclatura determinada, consistente en unas siglas y número de orden para cada uno (dos dígitos), separados por un guión. Se usará RUC para Requisito de Usuario de Capacidad, y RUR para Requisito de Usuario de Restricción.
- **Título:** Será conciso e unívoco.
- **Descripción:** será breve y no debe dar lugar a ningún tipo de ambigüedad.
- **Prioridad:** Determina la importancia del requisito. Tendremos tres niveles:
    - **Alta:** máxima importancia. Su desarrollo debe ser en un lugar prioritario.

    - **Media:** No son tan esenciales como los anteriores, pero deberán cumplirse siempre o comprometerán la funcionalidad y prestaciones del sistema
    - **Baja:** últimos en desarrollarse, no son vitales.
- **Necesidad:** define la utilidad que tiene para el proyecto. Pueden ser:
    - **Esencial:** De cumplimiento obligatorio.
    - **Deseable:** Aunque no es esencial, su correcta implementación aportaría gran calidad al sistema.
    - **Opcional:** Aconsejable su implementación, pero no son obligatorios.
- **Verificalidad:** Esta propiedad define el nivel de medición que tiene este requisito, en cuanto a si es posible comprobar su correcta implementación y en qué cantidad. Podrá ser de verificalidad alta, media o baja. Para requisitos de usuario, siempre tendrán una verificalidad alta.
- **Estabilidad:** Probabilidad en que este requisito se vea modificado en el medio o largo plazo.
    - **Alta:** no podrá ser modificado.
    - **Media:** Podrá ser modificado por cambios justificados en el proyecto.
    - **Baja:** Podrá ser modificado en distintas fases de realización del proyecto.
- **Fuente:** describe quién ha emitido el requisito. En nuestro caso, será siempre el alumno.

Por tanto, en base a lo anterior, tendremos la siguiente plantilla para cada uno de los requisitos:

Identificador:			
Título			
Descripción			
Prioridad	☐Alta	☐ Media	☐ Baja
Necesidad	☐ Esencial	☐ Deseable	☐ Opcional
Verificabilidad	☐ Alta	☐ Media	☐ Baja
Estabilidad	☐ Alta	☐ Media	☐ Baja
Fuente	Alumno		

Tabla 18 - Ejemplo de tabla de requisitos de usuario

Identificador: RUC-01	
Título	Registro aplicación usuario.
Descripción	El usuario podrá registrarse en la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-02	
Título	Autenticación en la aplicación usuario.
Descripción	El usuario podrá loggearse (autenticarse) en la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-03	
Título	Desconexión en la aplicación.
Descripción	El usuario podrá desconectarse (cerrar sesión) en la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-04	
Título	Consultar obra cercana.
Descripción	El usuario podrá consultar información relativa a la obra que tenga más cercana.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-05	
Título	Puntuar obra cercana.
Descripción	El usuario podrá puntuar la obra que tenga más cercana.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-06	
Título	No registro aplicación de administrador.
Descripción	El usuario no podrá registrarse en la aplicación de administrador.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-07	
Título	Autenticación aplicación administrador.
Descripción	El usuario-administrador podrá autenticarse en la aplicación de administrador del sistema del museo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-08	
Título	Desconexión aplicación administrador.
Descripción	El usuario-administrador podrá cerrar sesión en la aplicación de administrador del sistema del museo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-09	
Título	Consulta de obras vinculadas a beacons.
Descripción	El usuario-administrador podrá consultar las obras que se han dado de alta en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-10	
Título	Alta obras vinculadas a beacons.
Descripción	El usuario-administrador podrá dar de altas nuevas obras en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-11	
Título	Borrar obras vinculadas a beacons.
Descripción	El usuario-administrador podrá borrar obras en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-12	
Título	Actualizar obras vinculadas a beacons.
Descripción	El usuario-administrador podrá actualizar la información de obras en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-13	
Título	Consultar información completa obras vinculadas a beacons.
Descripción	El usuario-administrador podrá consultar la información completa relacionada con una obra en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUC-14	
Título	Consultar estadísticas de una obra vinculadas a beacons.
Descripción	El usuario-administrador podrá consultar las estadísticas de tiempo medio y el número de flujos a otras obras de una que esté vinculada a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

REQUISITOS DE RESTRICCIÓN

Identificador: RUR-01	
Título	Idioma
Descripción	El idioma empleado en la interfaz será el español
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUR-02	
Título	Conexión a internet
Descripción	Se requerirá conexión a internet durante todo el uso de la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUR-03	
Título	Conexión bluetooth
Descripción	Se requerirá tener activado el bluetooth del Smartphone durante todo el uso de la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

Identificador: RUR-04	
Título	Tipo de identificador del beacon
Descripción	En la aplicación de administrador, al dar de alta un beacon se requerirán un mínimo de 4 y un máximo de 5 caracteres, todos ellos numéricos.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	Alumno

3.4.3. CASOS DE USO

Una vez identificados los requisitos de usuario anteriores, podemos extraer los casos de uso necesarios para el correcto desarrollo y funcionamiento de la solución. Casos de uso serán aquellas acciones que los usuarios pueden realizar sobre nuestro sistema.

De la misma forma que se ha hecho en el apartado anterior, se usarán tablas que contendrán la definición de estos casos. A continuación, se definen los atributos que contendrán cada una de las tablas de cada caso de uso.

- **Caso de uso:** nombre del mismo, de forma concisa y unívoca, que lo define.
- **Actores:** Roles de los usuarios que pueden ejecutar el caso de uso.
- **Objetivo:** Explica la finalidad del caso de uso, sin ser ambiguo.
- **Precondiciones:** Estado en que se encuentra el sistema antes de la ejecución del caso de uso.
- **Postcondiciones:** Estado en el que se encuentra el sistema tras la ejecución del caso de uso.
- **Requisito con el que se relaciona:** Requisito de usuario definido en el apartado anterior y que está relacionado con ese caso de uso.

Tras esto, mostramos la plantilla que van a seguir las tablas de cada uno de los casos de uso:

Caso de uso:	
Actores	
Objetivo	
Precondiciones	
Postcondiciones	
Requisito relacionado	

Tabla 19 - Ejemplo de tabla de Casos de Uso

En el caso de la solución desarrollada en este trabajo de fin de grado, tendremos tres tipos de actores:

- **Usuario anónimo:** Usuario que no está autenticado en el sistema.
- **Usuario autenticado:** Usuario autenticado en el sistema.
- **Usuario-administrador autenticado:** Usuario autenticado en el sistema, con capacidades de administrador.

LISTADO DE CASOS DE USO

Caso de uso:	Registrarse
Actores	Usuario anónimo.
Objetivo	Obtener una cuenta de usuario con la que poder autenticarse.
Precondiciones	El usuario no está registrado en el sistema.
Postcondiciones	El usuario está registrado en el sistema.
Requisito relacionado	RUC-01

Caso de uso:	Autenticarse
Actores	Usuario anónimo.
Objetivo	Identificarse en el sistema para poder hacer uso del mismo como usuario autenticado.
Precondiciones	No está autenticado en el sistema.
Postcondiciones	Está autenticado en el sistema.
Requisito relacionado	RUC-02

Caso de uso:	Desconectarse
Actores	Usuario autenticado, usuario-administrador autenticado.
Objetivo	Dejar de estar autenticado en el sistema.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	No estar autenticado en el sistema.
Requisito relacionado	RUC-03

Caso de uso:	Consultar obra cercana.
Actores	Usuario autenticado, usuario-administrador autenticado.
Objetivo	Consultar la información relativa a la obra vinculada a beacon más cercana.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	Se muestra la información relativa a la obra vinculada a beacon más cercana.
Requisito relacionado	RUC-04

Caso de uso:	Puntuar obra cercana.
Actores	Usuario autenticado, usuario-administrador autenticado.
Objetivo	Puntuar de 0-10 la obra vinculada a beacon más cercana.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	Puntuar de 0-10 la obra vinculada a beacon más cercana.
Requisito relacionado	RUC-05

Caso de uso:	No registro en la aplicación de administrador
Actores	Usuario anónimo.
Objetivo	La aplicación de administrador debe no permitir el registro en la misma.
Precondiciones	No estar autenticado en el sistema.
Postcondiciones	No disponer de opción de registro
Requisito relacionado	RUC-06

Caso de uso:	Autenticarse aplicación administrador
Actores	Usuario anónimo.
Objetivo	Identificarse en el sistema para poder hacer uso del mismo como usuario autenticado.
Precondiciones	No está autenticado en el sistema.
Postcondiciones	Está autenticado en el sistema como usuario-administrador.
Requisito relacionado	RUC-07

Caso de uso:	Desconectarse aplicación administrador
Actores	Usuario-administrador autenticado.
Objetivo	Dejar de estar autenticado en el sistema.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	No estar autenticado en el sistema.
Requisito relacionado	RUC-08

Caso de uso:	Consultar obras vinculadas.
Actores	Usuario-administrador autenticado.
Objetivo	Consultar las obras que están dadas de alta en el sistema y están vinculadas, por tanto, a un beacon
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	Mostrará las obras dadas de alta en el sistema y que están, por tanto, vinculadas a un beacon.
Requisito relacionado	RUC-09

Caso de uso:	Dar de alta obras.
Actores	Usuario-administrador autenticado.
Objetivo	Dar de alta obras nuevas en el sistema, vinculándolas a un beacon y rellenando la información necesaria para mostrar a usuario.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	La nueva obra vinculada al beacon estará almacenada en el sistema y lista para ser mostrada.
Requisito relacionado	RUC-10

Caso de uso:	Borrar obras.
Actores	Usuario-administrador autenticado.
Objetivo	Borrar obras del sistema y que están, por tanto, vinculadas a un beacon.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	La obra dejará de estar almacenada en el sistema y no estará disponible para ser mostrada, y su número de beacon liberado.
Requisito relacionado	RUC-11

Caso de uso:	Actualizar obras.
Actores	Usuario-administrador autenticado.
Objetivo	Actualizar la información de una obra registrada en el sistema y que está, por tanto, vinculadas a un beacon.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	La información relacionada con la obra y que se muestra al usuario, se cambiará y quedará actualizada, lista para ser mostrada al usuario.
Requisito relacionado	RUC-12

Caso de uso:	Consultar información completa de una obra.
Actores	Usuario-administrador autenticado.
Objetivo	Poder consultar, de una obra concreta, toda la información que tiene relacionada.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	Se mostrará toda la información relativa a una obra concreta para su visualización.
Requisito relacionado	RUC-13

Caso de uso:	Consultar estadísticas de una obra.
Actores	Usuario-administrador autenticado.
Objetivo	Poder consultar, de una obra concreta, las estadísticas de visualización media de usuario y la cantidad de flujo a otras obras del sistema.
Precondiciones	Estar autenticado en el sistema.
Postcondiciones	Se mostrará para su visualización el tiempo medio de visualización de usuarios, así como una gráfica con la cantidad de flujos que hay de una obra a las demás.
Requisito relacionado	RUC-14

A continuación se muestra un diagrama resumen de los casos de uso, para poder identificar visualmente la interacción de cada rol de usuario con el sistema:

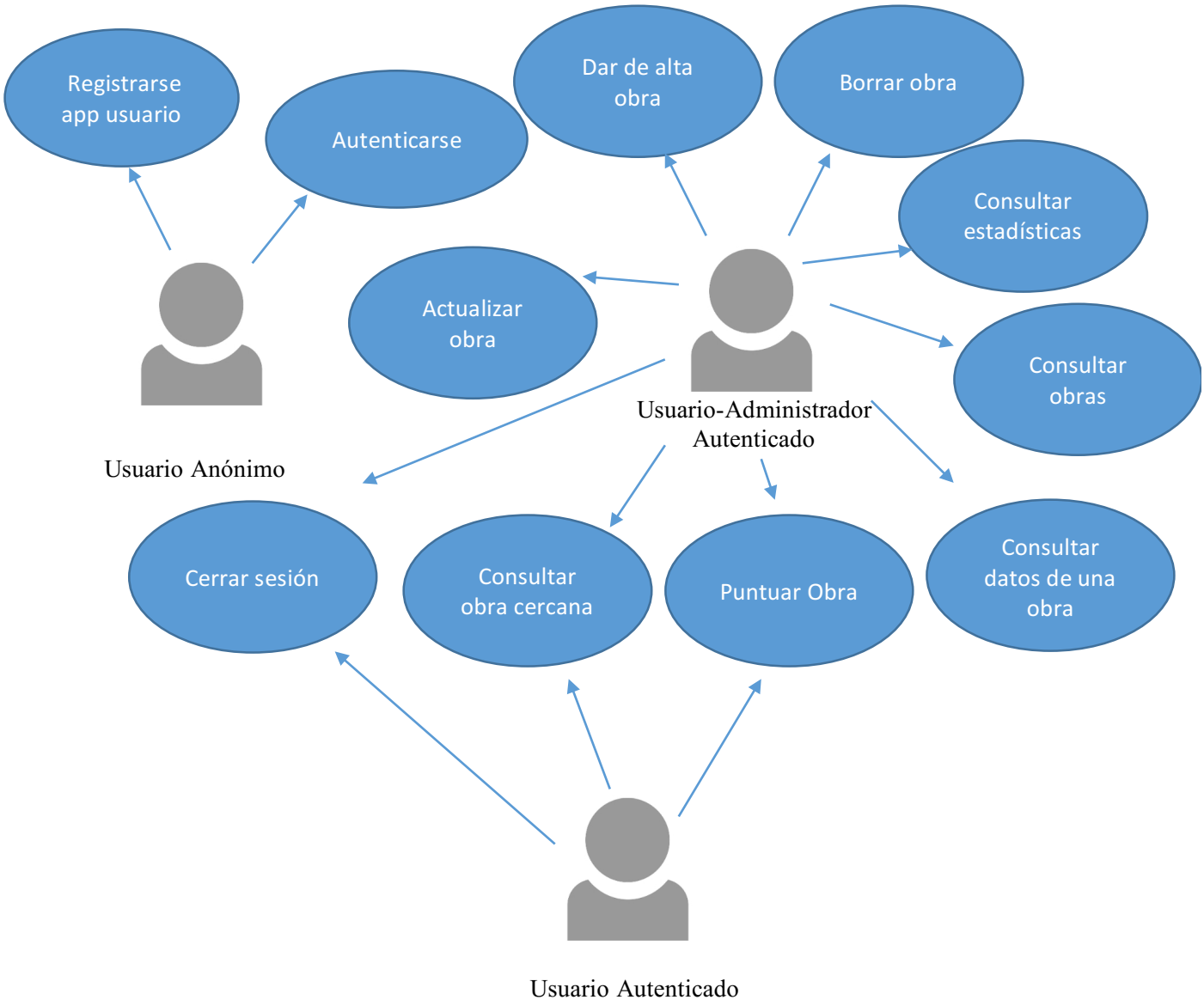


Ilustración 37 - Representación de Roles de Usuario y su interacción con el sistema

3.4.4. REQUISITOS DE SOFTWARE

Los requisitos de software definen qué debe hacer el sistema, sin profundizar en cómo debe hacerlo. A partir de los casos de uso y los requisitos de usuario, obtenemos los requisitos de software, que obviamente, el de nuestra solución debe cumplir.

Al igual que hemos hecho en los dos apartados anteriores, se seguirá un formato de tablas para identificar cada uno de estos requisitos. Los atributos de la misma serán similares a los que se muestran en los requisitos de usuario.

- **Identificador:** Debe ser unívoca y hacer referencia al requisito. Como se ha hecho en los requisitos de usuario, se seguirá una nomenclatura que consta de las siglas RSF para los Requisitos de Software Funcional, y RSR para Requisitos de Software de Rendimiento, seguido de un guion y dos cifras numéricas.
- **Título:** definirá el requisito, de forma concisa y unívoca.
- **Descripción:** No muy extensa y sin lugar a ambigüedades.
- **Prioridad:** Determina la importancia del requisito. Tendremos tres niveles:
    - **Alta:** máxima importancia. Su desarrollo debe ser en un lugar prioritario.
    - **Media:** No son tan esenciales como los anteriores, pero deberán cumplirse siempre o comprometerán la funcionalidad y prestaciones del sistema
    - **Baja:** últimos en desarrollarse, no son vitales.
- **Necesidad:** define la utilidad que tiene para el proyecto. Pueden ser:
    - **Esencial:** De cumplimiento obligatorio.
    - **Deseable:** Aunque no es esencial, su correcta implementación aportaría gran calidad al sistema.
    - **Opcional:** Aconsejable su implementación, pero no son obligatorios.
- **Verificalidad:** Esta propiedad define el nivel de medición que tiene este requisito, en cuanto a si es posible comprobar su correcta implementación y en qué cantidad. Podrá ser de verificalidad alta, media o baja. Para requisitos de usuario, siempre tendrán una verificalidad alta.
- **Estabilidad:** Probabilidad en que este requisito se vea modificado en el medio o largo plazo.
    - **Alta:** no podrá ser modificado.
    - **Media:** Podrá ser modificado por cambios justificados en el proyecto.
    - **Baja:** Podrá ser modificado en distintas fases de realización del proyecto.
- **Fuente:** Serán los requisitos de usuario de los que parte el requisito de software.

Estos requisitos tienen dos categorías principales: requisitos funcionales y no funcionales. Dentro de estos últimos, pueden ser, por el carácter de nuestra aplicación, de rendimiento, de interfaz, operación, documentación y seguridad.

REQUISITOS FUNCIONALES

Delimitan qué hace el sistema. Derivan de los casos de uso y de los requisitos de usuario de capacidad.

Identificador: RSF-01	
Título	Registro aplicación usuario.
Descripción	El sistema permitirá al usuario registrarse en la aplicación de usuario.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-01

Identificador: RSF-02	
Título	Autenticación en la aplicación usuario.
Descripción	El sistema permitirá al usuario loggearse (autenticarse) en la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-02

Identificador: RSF-03	
Título	Desconexión en la aplicación.
Descripción	El sistema permitirá al usuario desconectarse (cerrar sesión) en la aplicación.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-03

Identificador: RSF-04	
Título	Consultar obra cercana.
Descripción	El sistema permitirá usuario consultar información relativa a la obra que tenga más cercana.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-04

Identificador: RSF-05	
Título	Puntuar obra cercana.
Descripción	El sistema permitirá al usuario puntuar la obra que tenga más cercana.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-05

Identificador: RSF-06	
Título	No registro aplicación de administrador.
Descripción	El sistema no permitirá al usuario registrarse en la aplicación de administrador.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-06

Identificador: RSF-07	
Título	Autenticación aplicación administrador.
Descripción	El sistema permitirá al usuario-administrador autenticarse en la aplicación de administrador del sistema del museo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-07

Identificador: RSF-08	
Título	Desconexión aplicación administrador.
Descripción	El sistema permitirá al usuario-administrador cerrar sesión en la aplicación de administrador del sistema del museo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-08

Identificador: RSF-09	
Título	Consulta de obras vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador consultar las obras que se han dado de alta en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-09

Identificador: RSF-10	
Título	Alta obras vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador dar de altas nuevas obras en el sistema vinculadas a un beacon, se requerirán un mínimo de 4 y un máximo de 5 caracteres, todos ellos numéricos, para el identificador del beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-10, RUR-04

Identificador: RSF-11	
Título	Borrar obras vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador borrar obras en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-11

Identificador: RSF-12	
Título	Actualizar obras vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador actualizar la información de obras en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-12

Identificador: RSF-13	
Título	Consultar información completa obras vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador consultar la información completa relacionada con una obra en el sistema vinculadas a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-13

Identificador: RSF-14	
Título	Consultar estadísticas de una obra vinculadas a beacons.
Descripción	El sistema permitirá al usuario-administrador consultar las estadísticas de tiempo medio y el número de flujos a otras obras de una que esté vinculada a un beacon.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUC-14

REQUISITOS DE RENDIMIENTO

Para este tipo de requisitos, utilizaremos valores numéricos que cuantifican el rendimiento que ha de tener el sistema.

Identificador: RSR-01	
Título	Tiempo de reconocimiento de beacon.
Descripción	El software de la aplicación móvil deberá reconocer el beacon que tenga cercano y cargar la información relativa al cuadro que está vinculado a este en un tiempo máximo de 5 segundos.
Prioridad	☐Alta ■ Media ☐ Baja
Necesidad	☐ Esencial ■ Deseable ☐ Opcional
Verificabilidad	☐ Alta ■ Media ☐ Baja
Estabilidad	☐ Alta ■ Media ☐ Baja
Fuente	RUC-04

Identificador: RSR-02	
Título	Tiempo de peticiones HTTP.
Descripción	Las peticiones http lanzadas por la aplicación móvil, tanto de usuario como de administrador, tardarán un tiempo máximo de 5 segundos.
Prioridad	☐Alta ■ Media ☐ Baja
Necesidad	☐ Esencial ■ Deseable ☐ Opcional
Verificabilidad	☐ Alta ■ Media ☐ Baja
Estabilidad	☐ Alta ■ Media ☐ Baja
Fuente	RUC-1, RUC-2, RUC-4, RUC-5, RUC-7, RUC-9, RUC-10, RUC-11, RUC-12, RUC-13, RUC-14

REQUISITOS DE INTERFAZ

Estos requisitos describirán la forma en la que han de comunicarse los distintos módulos del sistema de software, tanto a nivel software como hardware.

Identificador: RSI-01	
Título	Almacenamiento de datos.
Descripción	Los datos se almacenarán en una base de datos que únicamente será accesible desde el servidor API-Rest.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	

Identificador: RSI-02	
Título	Gestión de peticiones.
Descripción	Todas las peticiones http realizadas por la aplicación de usuario y por la de administrador serán tratadas y procesadas por un API-Rest.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	

Identificador: RSI-03	
Título	Idioma de la interfaz.
Descripción	Tanto la interfaz de la aplicación de usuario como la interfaz de aplicación del administrador estará en español.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUR-01

REQUISITOS DE OPERACIÓN

Indican como se realizarán las tareas del sistema.

Identificador: RSO-01	
Título	Conexión a Internet.
Descripción	Se requerirá conexión a internet para todas las acciones que requieran envío o recepción de información con el servidor.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUR-02

Identificador: RSO-02	
Título	Conexión bluetooth activada.
Descripción	Se requerirá tener la conexión bluetooth del Smartphone activada para detectar los beacons.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	RUR-03

Identificador: RSO-03	
Título	Tiempo operativo.
Descripción	El sistema deberá estar operativo el 99% del tiempo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	☐ Alta ☐ Media ■ Baja
Estabilidad	☐ Alta ■ Media ☐ Baja
Fuente	

REQUISITOS DE DOCUMENTACIÓN

Especifican como se realiza la documentación del proyecto.

Identificador: RSD-01	
Título	Idioma del código fuente.
Descripción	El idioma del código fuente deberá ser en español mientras sea posible.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	

Identificador: RSD-02	
Título	Comentarios del código fuente.
Descripción	El código fuente deberá estar comentado en español, sobre todo aquellas partes que, por realizar trabajos de mayor complejidad, ayuden a entender el funcionamiento del mismo.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	

REQUISITOS DE SEGURIDAD

Describe los medios que se usarán para mantener un nivel de seguridad en el sistema, manteniendo confidencialidad e integridad del sistema.

Identificador: RSS-01	
Título	Contraseñas cifradas.
Descripción	Las contraseñas se enviarán siempre cifradas, y serán almacenadas de esta forma en el servidor. La comparación de contraseñas para la autenticación, de la misma forma, será mediante comparación cifrada, nunca descifrando la contraseña almacenada.
Prioridad	■Alta ☐ Media ☐ Baja
Necesidad	■ Esencial ☐ Deseable ☐ Opcional
Verificabilidad	■ Alta ☐ Media ☐ Baja
Estabilidad	■ Alta ☐ Media ☐ Baja
Fuente	

3.4.5. MATRIZ DE TRAZABILIDAD

A partir de la siguiente matriz, llamada de trazabilidad, podemos ver como interfieren los requisitos del software con los requisitos de usuario, y así ver como estos últimos quedan totalmente cubiertos.

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11	RUC-12	RUC-13	RUC-14	RUR-01	RUR-02	RUR-03	RUR-04
RSF-01	X																	
RSF-02		X																
RSF-03			X															
RSF-04				X														
RSF-05					X													
RSF-06						X												
RSF-07							X											
RSF-08								X										
RSF-09									X									
RSF-10										X								X
RSF-11											X							
RSF-12												X						
RSF-13													X					
RSF-14														X				
RSR-01				X														
RSR-02	X	X		X	X		X		X	X	X	X	X	X				
RSI-01																		
RSI-02																		
RSI-03															X			
RSO-01																X		
RSO-02																	X	
RSO-03																		
RSD-01																		
RSD-02																		
RSS-01																		

Tabla 20 - Matriz de trazabilidad

3.5. DISEÑO DEL SISTEMA

A partir del análisis realizado en el apartado anterior, sabemos cuál es el problema qué necesitamos resolver, teniéndolo perfectamente acotado y descrito. A continuación, se enunciará como resolverlo.

En primer lugar, se mostrará la **arquitectura del sistema** con el fin de tener una perspectiva general del mismo, el cual, como se ha comentado a lo largo de la presente memoria, el sistema no es un elemento único, sino varios que en su conjunto **forman una solución**.

3.5.1. ARQUITECTURA DEL SISTEMA

Ilustración 38 - Arquitectura del Sistema

Tal y como podemos observar en el esquema anterior, el sistema de la solución planteada consta de cuatro capas. En la capa inferior, tenemos los beacons. Estos dispositivos simplemente emitirán una señal para "quien la quiera captar", enviando información de quién es: identificador UUID, Major Number y **Minor Number**.

En la capa inmediatamente superior, tenemos la parte del sistema con la que se relaciona el usuario. Esta capa, actua como cliente de la capa que tiene por encima, el servidor API-Rest, basado en el paradigma cliente-servidor.

La capa del Servidor API-Rest, que es la siguiente, es la que actúa como **controlador**, dado que gestiona todas las peticiones http recibidas y actúa en consecuencia, obteniendo o registrando datos en la base de datos, y recibiendo y enviando estos al cliente, que como hemos comentado, con aplicaciones móviles. Además, este servidor tiene la particularidad que, al trabajar con una base de datos **No-SQL**, gestiona también el modelo de los datos que se almacenan en ella, puesto que **MongoDB** como base de datos not-only-SQL de tipo documental, aceptaría cualquier formato **JSON** que se le enviara. No obstante, para poder tratarlos, necesitamos que tengan un pequeño orden.

En la última capa, y, por tanto, la más superior, tenemos la base de datos. Contiene toda la información de los usuarios, por un lado, y de las obras expuestas, por otro.

3.5.2. SUBSISTEMAS

En este apartado se analiza cada uno de los subsistemas explicado anteriormente en forma de capas.

BASE DE DATOS

Como se ha explicado en el apartado anterior, la base de datos de nuestro sistema contiene, por un lado, la información de todos los usuarios y de las obras que están vinculadas a algún beacon y, por tanto, son localizables por nuestro sistema.

En cuanto a las tablas que lo conforman, al no ser una base de datos relacional al uso, se podría haber incluido incluso todos los datos en la misma tabla, indistintamente de la longitud, los distintos campos que tuvieran los distintos datos, etc. Porque, recordando, las bases de datos NoSQL aceptan una ristra de datos sin más. Pero se debe tener en cuenta, que por qué por el que se eligieron las bases de datos No-SQL no era esta capacidad de absorber cualquier tipo de dato, como en otros entornos si puede ser de utilidad, sino por la velocidad a la hora de tratarlos. Por tanto, en lo que respecta a la base de datos, se han creado dos tablas:

- **cuadros:** contiene toda la información de las obras que se den de alta en el sistema. Además de los datos del modelo de obras que gestiona el servidor, añade un identificador único y unívoco a cada documento JSON introducido.
- **users:** en este caso contiene toda la información de los usuarios dados de alta en el sistema. De la misma forma, a los atributos del modelo de usuario que controla el servidor, le añade un identificador único.

SERVIDOR API-REST

El servidor será programado en Javascript, en un entorno de ejecución llamado Node.js. El servidor se encarga básicamente de gestionar, en primer lugar, las rutas de las peticiones HTTP recibidas, que serán conducidas a las funciones que en cada caso sean oportunas, para tratar la petición y los datos que contengan.

```
//Devuelve todos los cuadros almacenados en mongodb
api.get('/cuadro', cuadroCtrl.getCuadros)
//Devuelve cuadro en funcion de su ID
api.get('/cuadro/:cuadroID', cuadroCtrl.getCuadro)
api.post('/cuadro', cuadroCtrl.saveCuadros)
api.put('/cuadro/:cuadroID',  cuadroCtrl.updateCuadro)
api.delete('/cuadro/:cuadroID', cuadroCtrl.deleteCuadro)
//Rutas de tipo post para usuario
api.post('/signup', userCtrl.signUp)
api.post('/signin', userCtrl.signIn)
api.put('/actualizar/:userID', userCtrl.updateUser)
api.get('/1/:userID', userCtrl.getUser)
```

Ilustración 39 - Rutas del API-Rest

Como podemos observar, tenemos las distintas extensiones que tendrá la URL del lugar donde se aloje nuestro servidor API-Rest, y cada una de ellas, es dirigida a una función para tratar cada uno de los datos. Tenemos las siguientes rutas:

- Tipo GET, para realizar una petición de datos:
    - **/cuadro:** devuelve todas las obras almacenadas en la base de datos a través de la función getCuadros, en formato JSON, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola..

- **/cuadro/:cuadroID:** llama a la función getCuadro, que devuelve la obra que encaje con el minor number que se ha pasado por el parámetro cuadroID, en formato JSON, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola..
- **/1/:userID:** llama a la función getUser, que devuelve el usuario que coincida con el mail userID pasado por parámetro, en formato JSON, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola..

- Tipo POST, se envían datos al servidor para que este haga un tratamiento de los mismos:
  - **/cuadro:** llama a la función saveCuadros. Este método tratará el cuerpo de la petición HTTP enviada, que contendrá los atributos que se deben introducir para crear un nuevo cuadro. Se encargará de crear un objeto Cuadro que tendrá como atributos los declarados en el modelo de las obras, que se explicarán más adelante, y serán guardados en la base de datos en formato JSON, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola.
  - **/signup:** llamará a la función signUp, la cual tratará los datos enviados en el cuerpo de la petición HTTP. Se hará en base al modelo de usuarios, encriptará la contraseña, y se almacenará así en la base de datos. De esta forma, un usuario nuevo quedará dado de alta, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola.
  - **/signin:** llama a la función signIn. En este caso, esta función hará una consulta a la base de datos buscando un usuario que coincida con el mail enviado en el body de la petición. Si se encuentra, hará una comparación de encriptación entre la clave enviada y la almacenada. Si coinciden, mandará una respuesta satisfactoria y se permitirá al usuario usar la aplicación. En caso de error, lo notificará y lo imprimirá por consola.
- Tipo PUT, se envían datos al servidor para que este actualice datos de la base de datos:
  - **/cuadro/:cuadroID:** llama a la función updateCuadro. Esta función se encargará de buscar el minor number de la obra almacenada en la base de datos que coincide con el pasado como parámetro en la petición. Una vez localizado, actualizará los nuevos datos en esa obra en la base de datos, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola.
  - **/actualizar/:userID:** llama a la función updateUser. Trata los datos de manera similar al anterior, solo que buscará un usuario en la tabla users cuyo mail coincida con el pasado por parámetro en la petición HTTP, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola.
- Tipo DELETE, para borrar datos de la base de datos:
  - **/cuadro/:cuadroID:** llama a la función deleteCuadro. Se encargará de borrar de la base de datos la obra almacenada cuyo minor number coincida con el pasado por parámetro, y mandará un mensaje de Ok. En caso de error, lo notificará y lo imprimirá por consola.

A continuación, se muestra una tabla de las distintas rutas a las que cada una de las aplicaciones mandará peticiones.

	APLICACIÓN USUARIO	APLICACIÓN USUARIO-ADMINISTRADOR
GET: /cuadro	✓	✓
GET: /cuadro/:cuadroID	✓	✓
GET: /1/:userID	X	✓
POST: /cuadro	X	✓
POST: /signup	✓	X
POST: /signin	✓	✓
PUT: /cuadro/:cuadroID	✓	✓
PUT: /actualizar/:userID	✓	X
DELETE: /cuadro/:cuadroID	X	✓

Tabla 21 - Rutas usadas por cada aplicación

A modo de resumen, a continuación, se muestra una tabla con la información enviada a cada petición http y qué se espera recibir de ella.

	INFORMACIÓN ENVIADA	RESPUESTA
GET: /cuadro	-	JSON con las obras almacenadas
GET: /cuadro/:cuadroID	Minor Number por parámetro	JSON con la obra almacenada que tiene asociado el Minor Number enviado por parámetro.
GET: /1/:userID	Dirección mail por parámetro	JSON con la información cifrada del usuario cuyo mail es el enviado por parámetro.
POST: /cuadro	Datos necesarios del modelo de cuadro	OK/ERROR
POST: /signup	Mail y contraseña	OK/ERROR
POST: /signin	Mail y contraseña	OK/ERROR
PUT: /cuadro/:cuadroID	En el Body datos que se deseen actualizar de la obra cuyo Minor Number de Beacon asociado se manda por parámetro.	OK/ERROR

PUT: /actualizar/:userID	Datos que se deseen actualizar del usuario cuyo mail coincide con el que se manda por parámetro	OK/ERROR
DELETE: /cuadro/:cuadroID	Beacon asociado a la obra que deseamos borrar por parámetro.	OK/ERROR

Tabla 22 - Información enviada y respuesta según petición

Dado la naturaleza de nuestro servidor, y que los modelos de los datos no son tratados por la base de datos sino por el mismo, es necesario mostrar los pertenecientes a las obras y usuarios almacenados.

```
const UserSchema = new Schema({

  email: { type: String, unique: true, lowercase: true},
  displayName: String,
  password: String,
  signupDate: {type: Date, default: Date.now()},
  admin: {type: String, default: 'false'},
  minorNumberFavorito: String,
  puntuacion: String,
  tiempo: String

})
```

Ilustración 40 - Modelo de Usuario

En el modelo de usuario, vemos que tendremos un email, una contraseña, una fecha de alta en el sistema, un atributo 'admin' que será true únicamente para el administrador y por defecto será false para el resto de usuario que se den de alta en el sistema, y por último, para gestionar las obras favoritas de este usuario, el Minor Number del beacon que está vinculado a la obra que mejor ha puntuado y ha estado más tiempo delante de ella consultando su información.

```
const CuadroSchema = Schema ({
  nombre: String,
  imagen: String,
  anyo: String,
  autor: String,
  tipo: {type: String, enum: ['Arquitectura', 'Escultura', 'Pintura']},
  categoria: String,
  descripcion: String,
  contadorMinorNumber12487: {type: String, default: '0'},
  contadorMinorNumber9278: {type: String, default: '0'},
  contadorMinorNumber57569: {type: String, default: '0'},
  contadorMinorNumber34637: {type: String, default: '0'},
  contadorMinorNumber35194: {type: String, default: '0'},
  contadorMinorNumber41145: {type: String, default: '0'},
  beaconMinorNumber: String,
  urlCupon: String,
  visualizaciones: {type: String, default: '0'},
  tiempoVisualizacion:{type: String, default: '0'}
})
```

Ilustración 41 - Modelo de Obra

Para el caso de modelo de obras, tenemos un nombre para la obra, una url de la imagen, un autor, un tipo de obra que está acotado, en este caso, a arquitectura, escultura y pintura; una categoría, pudiendo ser renacentista, gótica, etc.; una descripción, una relación de las veces que se ha pasado de esta obra a otra, mediante un contador que se incrementa cuando se produce un **flujo de una zona a otra**, delimitada por el punto en el que un beacon está más cercano que otro, una url del **cupón de descuento** vinculado a esa obra, el número de visualizaciones que tiene esa obra, y el tiempo que ha habido usuarios visualizando esta obra. Obviamente, el más importante, es el minor number del beacon que está relacionado con esta obra.

Por último, con el fin de explicar el funcionamiento, de manera gráfica, del servidor, se muestra la siguiente ilustración:

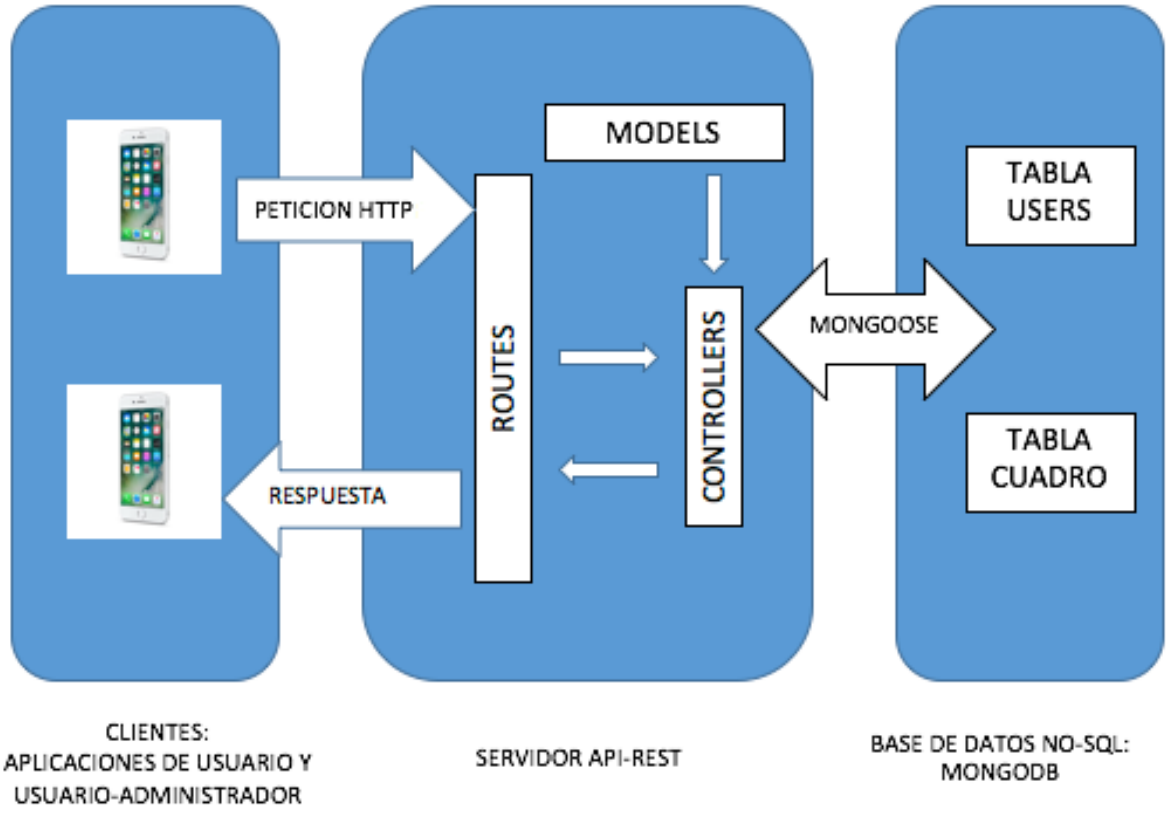


Ilustración 42 - Funcionamiento general del sistema

APLICACIONES CLIENTES: APP USUARIO Y APP USUARIO-ADMINISTRADOR

Aprovechando la capacidad computacional que tienen actualmente los smartphones, todo el modelo de vistas de las aplicaciones tanto de usuario como de usuario-administrador será generado por sí mismo. La única ayuda exterior que necesita es la comunicación con los beacons, **recibiendo** sus señales bluetooth e **interpretándolas**, y con el servidor API-Rest, al que enviará una serie de peticiones y esperará una determinada respuesta sobre la que actuar.

A continuación, se mostrará el diagrama de flujo de cada uno de las dos aplicaciones acompañadas del diagrama generado por el propio xCode, entorno donde se han desarrollado las aplicaciones móviles en lenguaje **Swift** para smartphones con sistema operativo iOS.

APLICACIÓN USUARIO

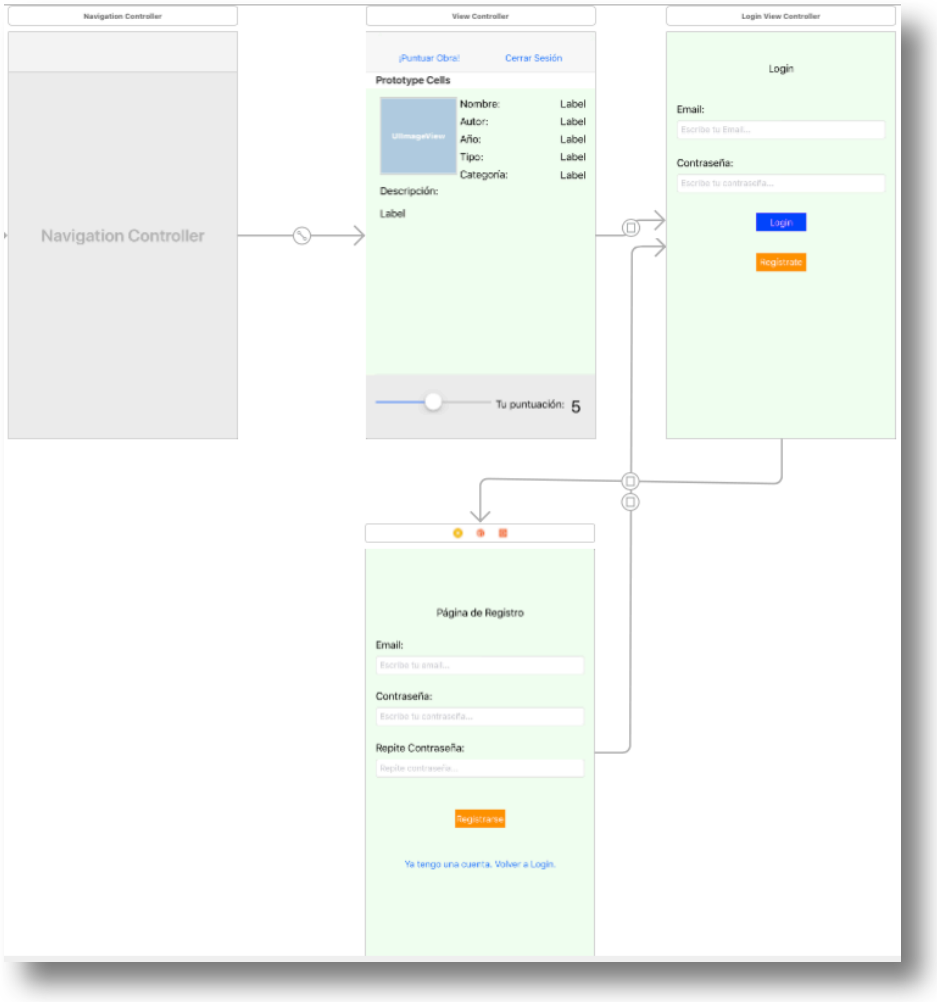


Ilustración 43 - StoryBoard de la aplicación de usuario

En la imagen anterior, podemos observar tres vistas, arriba a la derecha la del login, que permitirá, en caso de estar registrado y autenticarse el usuario correctamente, la visualización de la página principal. Si no estuviese registrado, podrá acceder a una vista donde hacerlo.

El diagrama de flujo de esta aplicación será el siguiente:

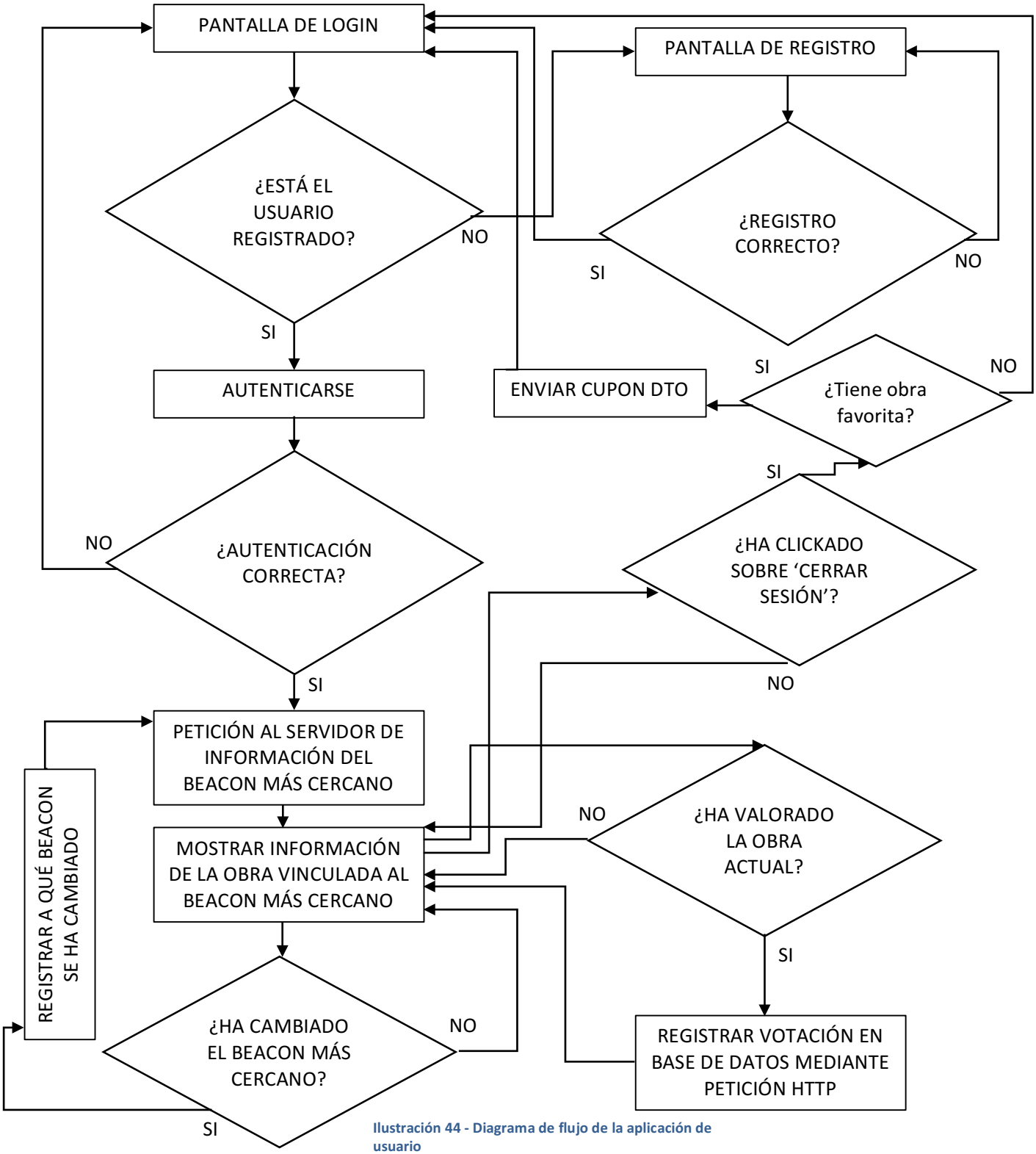


Ilustración 44 - Diagrama de flujo de la aplicación de usuario

Como podemos observar en el diagrama anterior, al iniciar la aplicación de usuario lo primero que se mostrará será un formulario para realizar el login y permitir al usuario autenticarse. En el caso de no tener cuenta, se puede acceder a un formulario de registro, para posteriormente volver a la pantalla de login y, ahora sí, el usuario se pueda **autenticar**.

Una vez autenticado, se mostrará la información del beacon más cercano, que está **vinculado a una obra** en concreto. Si no se realiza ninguna acción, permanecerá en ese estado. Si cambiamos a otra zona y se detecta que el beacon más cercano ha cambiado, **cambiará la información** que se muestra al usuario. En el caso de valorar una obra, aparecerá un mensaje satisfactorio y se seguirá mostrando la información de la obra vinculada al becon más cercano al usuario.

APLICACIÓN USUARIO-ADMINISTRADOR

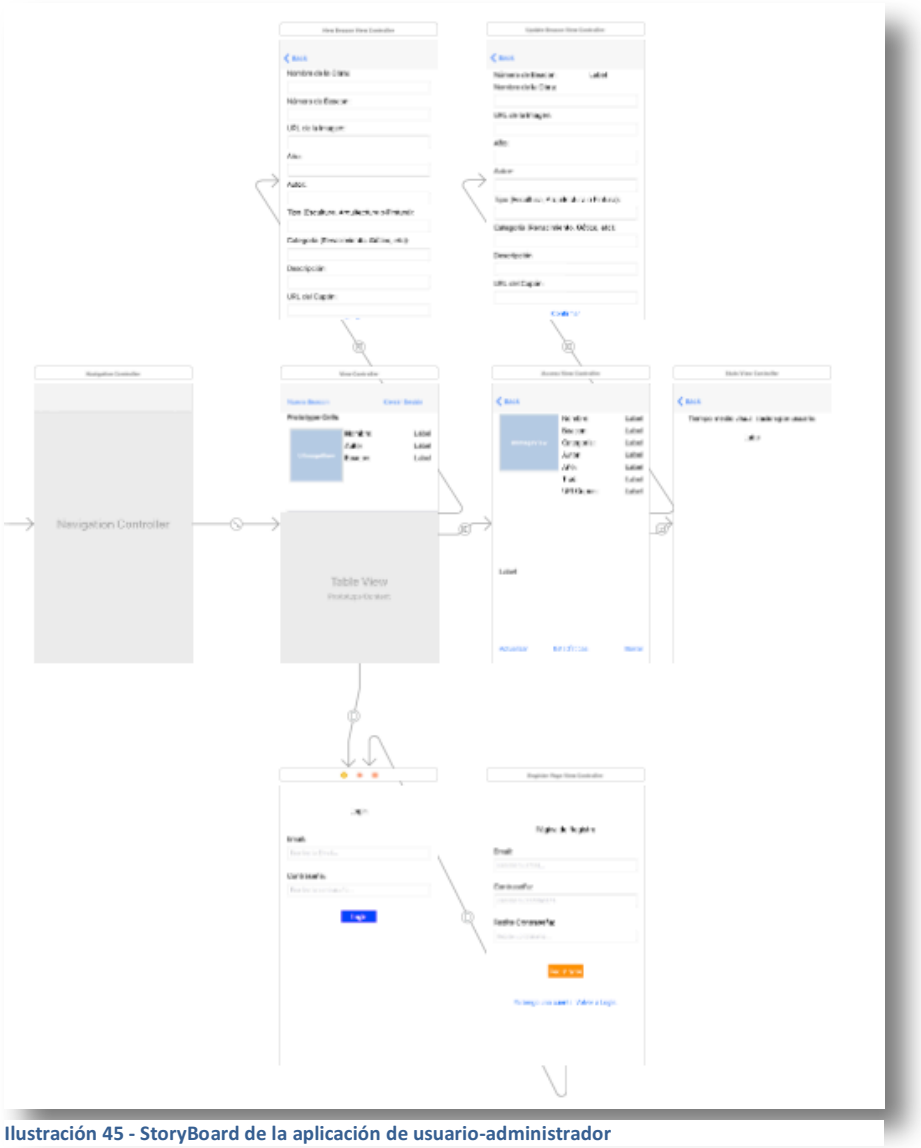


Ilustración 45 - StoryBoard de la aplicación de usuario-administrador

Para el caso de la aplicación de usuario-administrado, tendremos una pantalla inicial de login, que, tras autenticarnos, nos permitirá llevar a la página principal donde se muestran **las obras actualmente dadas de alta en el sistema**. Desde esta pantalla podemos seleccionar una de las obras, vincular un nuevo beacon, o cerrar sesión. Si realizamos esto último, pasaremos de nuevo a la pantalla de login y nuestra sesión quedará cerrada. Si clicamos sobre la acción para crear un nuevo beacon, seremos reconducidos a un formulario con los datos necesarios para rellenar para dar de alta la nueva obra en el sistema, y si hacemos click sobre una obra de la lista que aparece en la pantalla inicial, seremos reconducidos a una pantalla con una vista detallada y más extensa de esta obra. En ella podremos actualizar sus datos, borrarla, o ver sus estadísticas. Si nuestra elección es esta última, veremos una pantalla con el tiempo medio que un usuario está delante de este beacon, y mostrará una gráfica con la cantidad de transiciones que tiene a otros beacon, para así estudiar la colocación efectiva de las obras en función del flujo de usuarios, si alguno de estos flujos, por el recorrido planificado, no tuviese el sentido que debería. Si eligiéramos la opción de actualizar, seríamos reconducidos a un formulario donde podremos rellenar los campos que queremos actualizar.

Para este caso, considero que no es necesario una representación mediante un diagrama de flujo puesto que es bastante lineal y no dispone, como en la aplicación de usuario, de condiciones que pueden cambiar por el movimiento de este. En el caso de la aplicación de usuario-administrador, una vez se realiza el login, todo dependerá de una navegación lineal en función de lo que se elija en cada momento, como veremos en el siguiente apartado.

3.5.3. INTERFACES DE USUARIO

Los distintos roles de usuario interactuarán con el sistema a través de dos aplicaciones móviles, que, como hemos explicado, darán acceso a unos u otros según su rol. A continuación, se mostrarán las distintas interfaces de usuario de estas aplicaciones móviles.

APLIACIÓN MÓVIL DE USUARIO

Así como en el apartado anterior se han tratado el **flujo y el diseño de la aplicación** de usuario, en este apartado se va a mostrar la interfaz que se muestra al usuario y que dará al usuario la posibilidad de utilizar las distintas funcionalidades que tiene a su alcance.

En el caso de esta aplicación, no tiene más navegabilidad que el login, la pantalla de registro, y la que se muestra una vez autenticado, por lo que no tiene botones de navegación más allá que autenticarse, pasar de login a registro y volver, y cerrar sesión en la pantalla principal.

Como se comentaba anteriormente, en la primera pantalla el usuario no está autenticado. Como se puede observar en la captura que se muestra a continuación, tiene la posibilidad de autenticarse escribiendo su mail y contraseña y pulsando el botón de 'Login'. Si no tuviera una cuenta registrada en el sistema, puede acceder a la pantalla de registro pulsando en el botón 'Regístrate'.

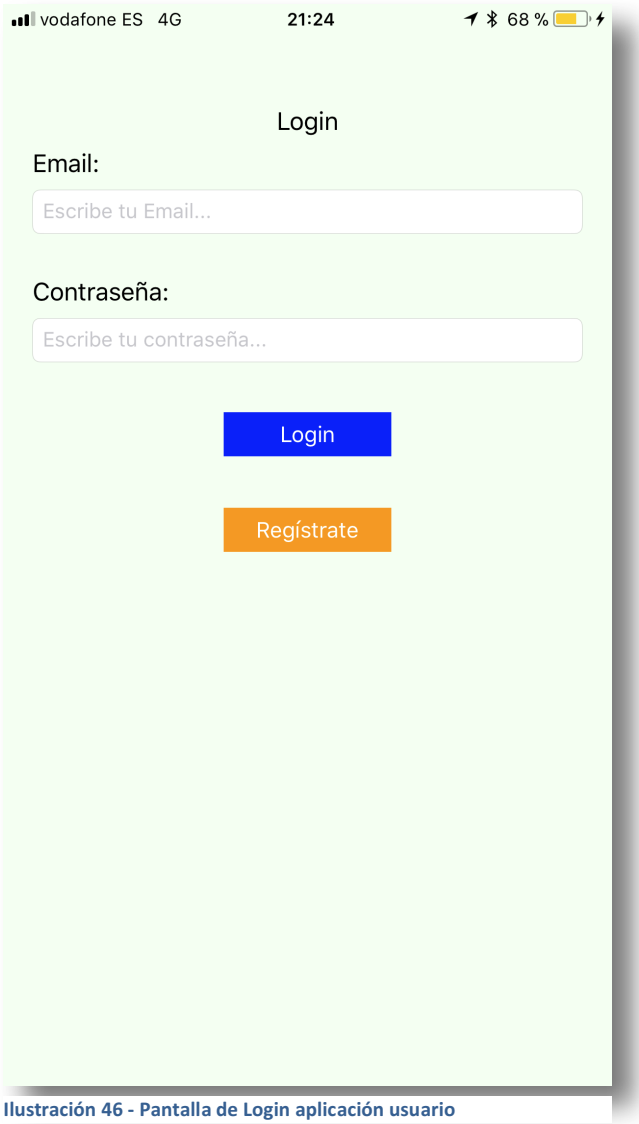


Ilustración 46 - Pantalla de Login aplicación usuario

Si pulsamo el botón de registro, accederemos a la página que se muestra bajo este texto. En la misma, se puede observar un formulario similar al del Login, con la particularidad de tener que repetir la contraseña para evitar errores de escritura y que, posteriormente, la cuenta no sea accesible. Una vez rellenados los campos, con el botón 'Registrarse', enviarán los datos al servidor y serán almacenados en la base de datos. Tras esto, seremos reconducidos a la página anterior. De la misma manera, si se quisiera volver a la pantalla de Login sin registrarse, podemos usar el botón 'Ya tengo una cuenta. Volver a Login'.

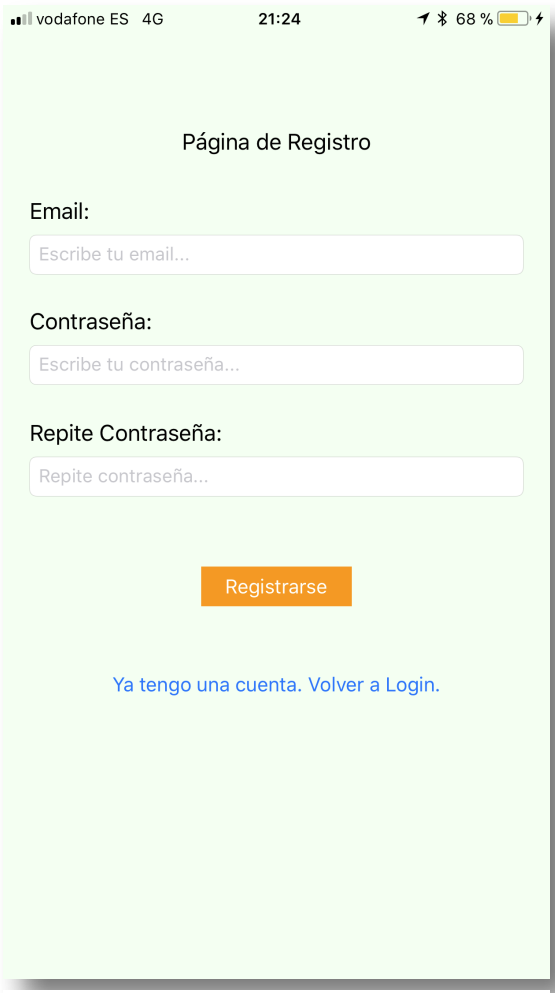


Ilustración 47 - Pantalla de Registro aplicación de usuario

Una vez autenticado, el usuario accederá a la **información de la obra** que esté vinculada al beacon más cercana al Smartphone del usuario, y, por tanto, a este. En dicha pantalla puede ver una foto de la obra, el nombre del autor, etc. Además, en la parte inferior tenemos un botón deslizable con el que podemos establecer un valor para esta obra que estamos visualizando. Una vez que queramos puntuar la obra, tenemos el botón de confirmación en la parte superior izquierda. Al lado de este botón, tenemos el de cierre de sesión, que nos llevará de nuevo a la pantalla de login.

Ilustración 48 - Pantalla principal aplicación de usuario

En el momento de cerrar sesión, si el usuario ha puntuado una obra, en la que además ha estado un tiempo determinado observándola, y cumple una regla de negocio de ejemplo preestablecida que si los segundo que está delante de la obra multiplicado por la puntuación que le dé es mayor que 100, se otorgará al usuario un cupón de un 25% de descuento en la tienda de souvenirs de artículos relacionados con esa obra. De esta forma, **maximizaremos las ventas** de este tipo de artículos, dando además al usuario un trato personalizado y un marketing dirigido en función de sus gustos.

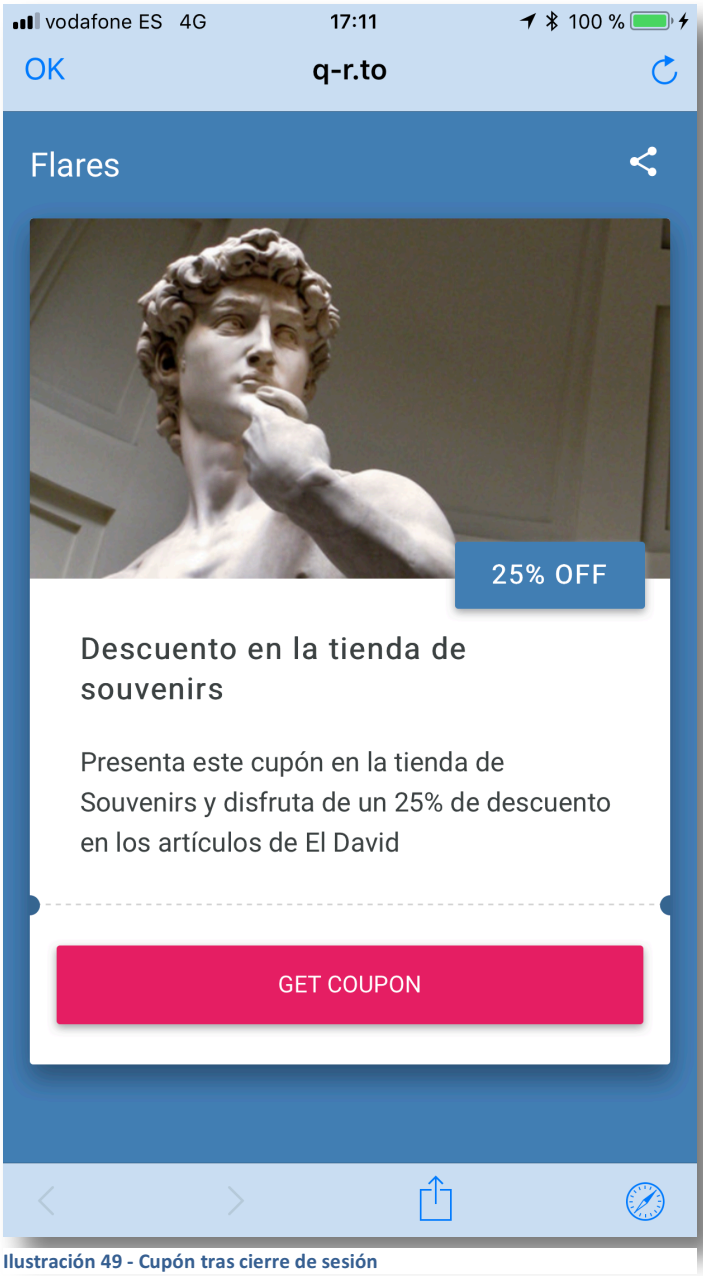


Ilustración 49 - Cupón tras cierre de sesión

APLICACIÓN MÓVIL USUARIO-ADMINISTRADOR

Para el caso de la aplicación móvil que usará el administrador del sistema del museo, tendremos una serie de cambios con respecto a la aplicación de usuario como visitante del museo. Uno de los principales es, que al tener una mayor opción de pantallas en las que navegar, se ha tenido que implementar botones para volver a interfaces anteriores durante la navegación.

Al igual que en la aplicación de usuario, en primer lugar, se mostrará una **interfaz de acceso** tipo login. En este caso, únicamente tendremos la posibilidad de introducir un mail y una contraseña, y podremos presionar sobre el botón 'Login', dando acceso únicamente a aquel usuario que tenga en el modelo de usuario almacenado en la base de datos el atributo admin con valor true. No se permite una ventana de registro puesto que esto comprometería la seguridad del sistema. Si bien no es posible poner el **atributo admin** en la base de datos en valor true nada más que manualmente, directamente se quita también esa pantalla por no ser útil y, como decíamos, por seguridad.

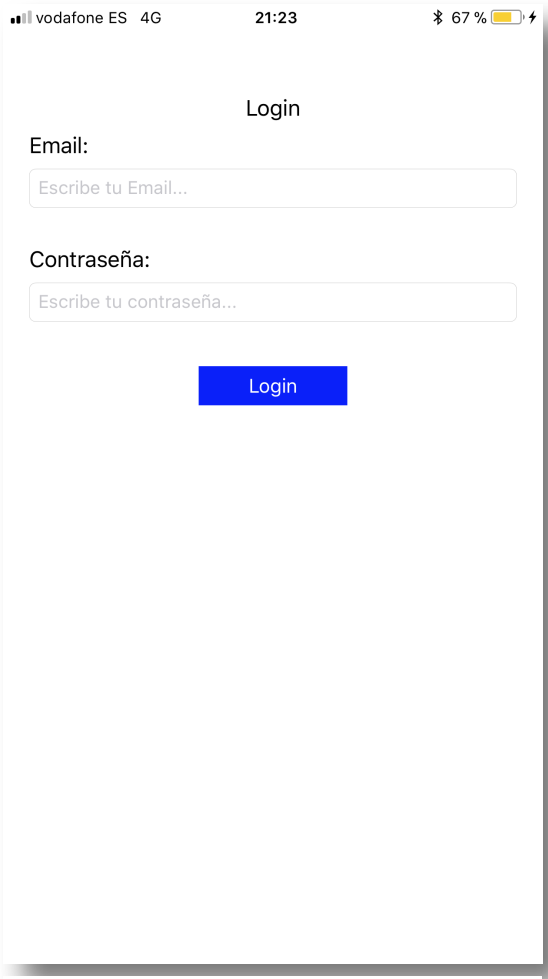


Ilustración 50 - Pantalla de Login aplicación de usuario-administrador

Una vez se ha autenticado el usuario-administrador, se llega a la pantalla de inicio donde se muestra, en primer lugar, una lista con información resumida de todas las obras dadas de alta en el sistema. Además, tenemos la posibilidad de pinchar en cada una de ellas, **vincular una obra nueva** a un nuevo beacon con el botón 'Nuevo Beacon' en la parte superior izquierda, y cerrar sesión en la parte superior derecha.

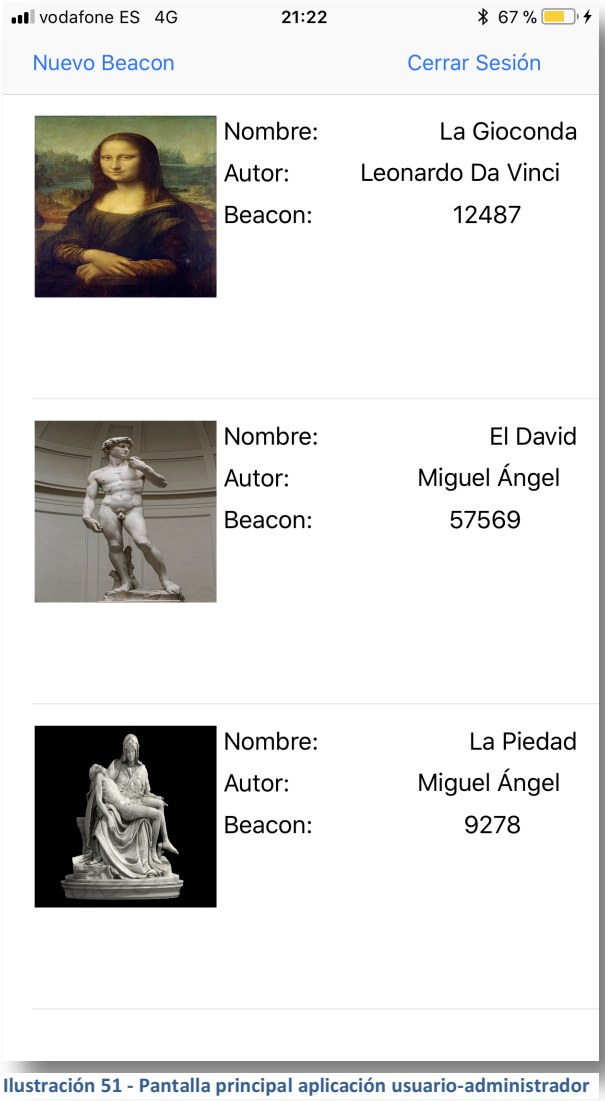


Ilustración 51 - Pantalla principal aplicación usuario-administrador

En el caso de clicar sobre el botón ‘Nuevo Beacon’, se conducirá al usuario de la aplicación a un formulario con todos los datos a rellenar para poder dar de alta un nuevo beacon y vincularlo a una obra del museo. Como hemos señalado anteriormente, en este caso incluimos un botón de ‘Volver’ para facilitar la **navegabilidad** en la aplicación.

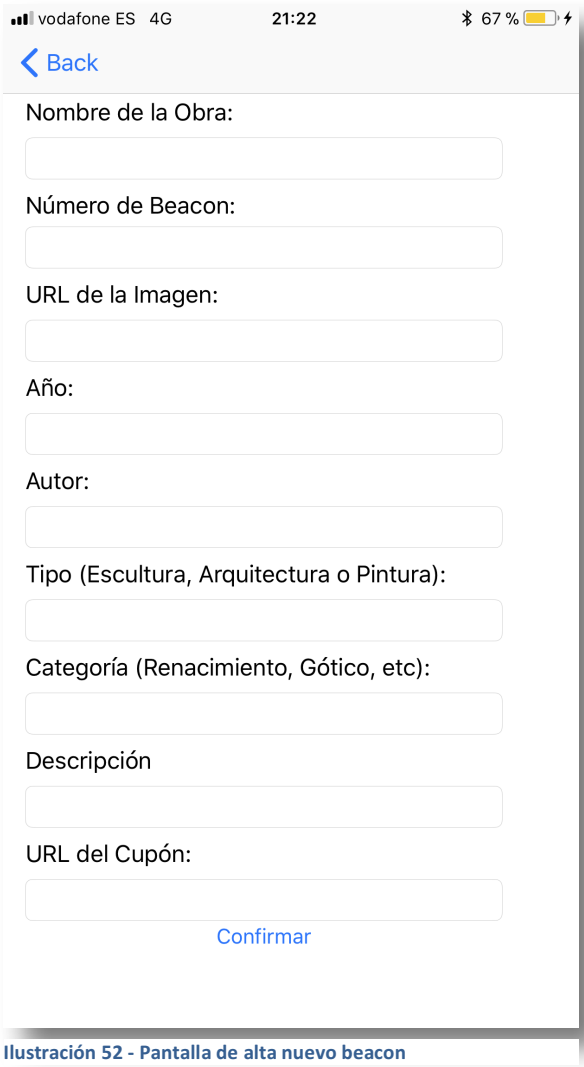


Ilustración 52 - Pantalla de alta nuevo beacon

En el caso de clickar sobre alguna de las obras que aparece en la pantalla principal en forma de lista, seremos dirigidos a una pantalla en la que podremos consultar la información completa de la misma. Además, tenemos la posibilidad de **actualizar la misma, borrarla, o consultar las estadísticas** de la misma, con botones en la parte inferior de la pantalla.

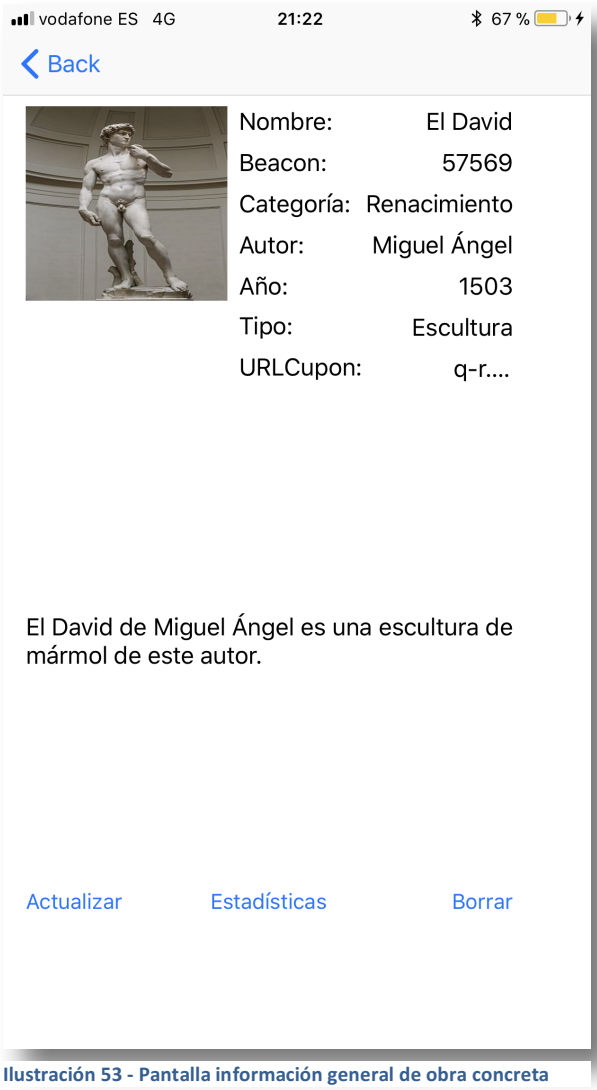


Ilustración 53 - Pantalla información general de obra concreta

Si el usuario clicara sobre el botón de actualizar, sería redirigido a la pantalla de actualización del beacon relacionado con la obra. En este caso, al contrario que el formulario de crear nuevo beacon, el usuario puede **rellenar el campo que** desee y este será actualizado en la base de datos y estará disponible para el uso del sistema desde ese preciso momento.

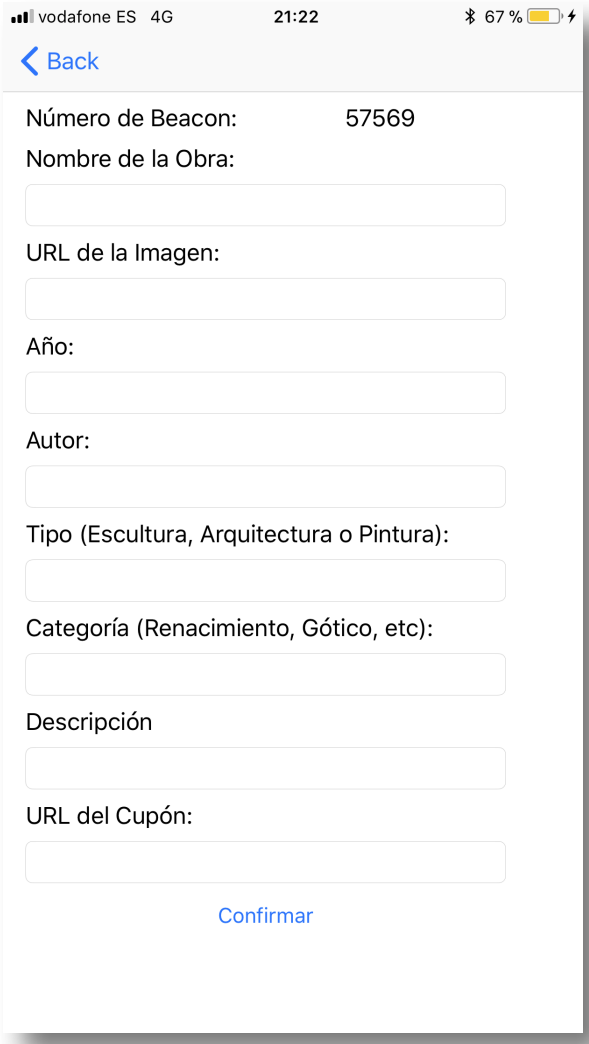


Ilustración 54 - Pantalla actualización beacon

En cambio, si el usuario-adminsitrador decide clicar sobre el botón de estadísticas, accederá a la información del **tiempo medio** de visualización por usuario, y de las **transiciones** desde el beacon actual al resto.

Como podemos observar, en la captura de pantalla que se muestra, el tiempo medio para el beacon 57569 es de 1,7 segundos por usuario. En la gráfica podemos observar que al beacon 9278 desde el beacon actual el número de transiciones es de cerca de 80, y el segundo beacon al que se suele transitar es el 35194, con alrededor de 60 transiciones. Entendemos que esto sucede porque estas tres obras están cerca o en la misma sala. Este gráfico sirve para estudiar como fluyen los usuarios a lo largo de su visita y extraer si la situación de las obras es la adecuada en función del comportamiento de los usuarios. Por ejemplo, si una de estas obras no estuviera en la misma sala que el beacon actual y obtuviese muchas transiciones, y otra que si lo está no las tuviese, es señal que alguna de las obras debería ser reubicada para mejor experiencia de usuario.

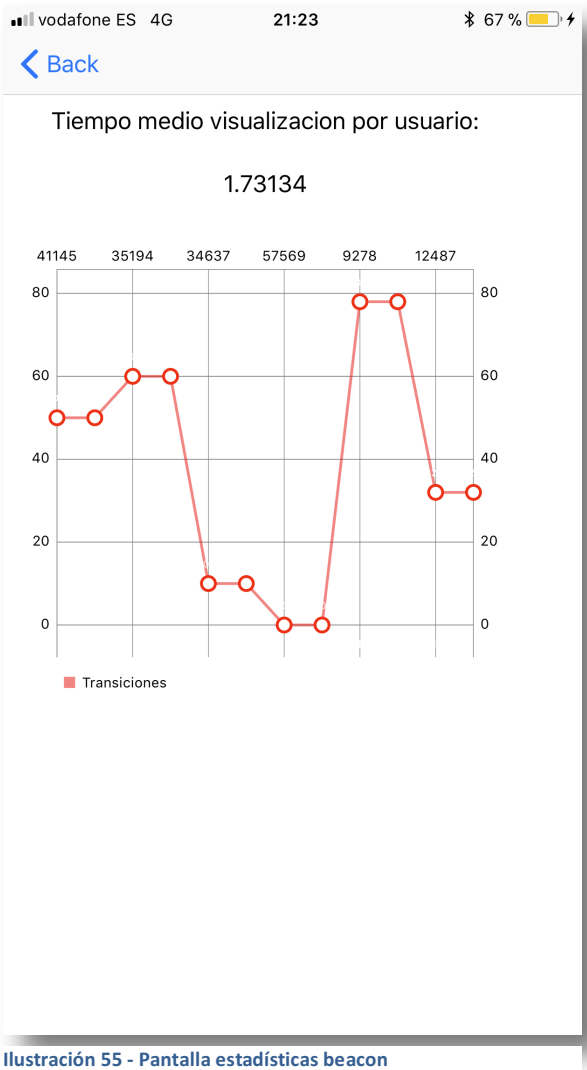


Ilustración 55 - Pantalla estadísticas beacon

3.5.4. EJEMPLO DE USO

En el presente apartado, se muestra un ejemplo de uso por cada uno de las aplicaciones que tenemos en el sistema, cubriendo las principales acciones que se pueden realizar en el mismo. Para ello, se creará una situación imaginaria y se describirá paso por paso.

Fernando, tras varios años gestionando las operaciones del museo en el que trabaja, le han encargado administrar un nuevo sistema implantado en el museo que permite digitalizar el museo, de forma que los visitantes, en su propio móvil, pueden consultar información de las obras que admiran y extraer datos de su comportamiento en el interior. Además, permitirá mejorar la fidelización de los visitantes, maximizar las ventas de la tienda de souvenirs, etc. Tras leerse el libro de instrucciones y recibir los beacons que se utilizarán, comienza, en primer lugar, a dar de alta obras en el sistema tras autenticarse con la cuenta de administrador que han puesto a su disposición. Tras dar de alta la primera obra, se dispone a recargar la pantalla principal para ver si aparece en ella.

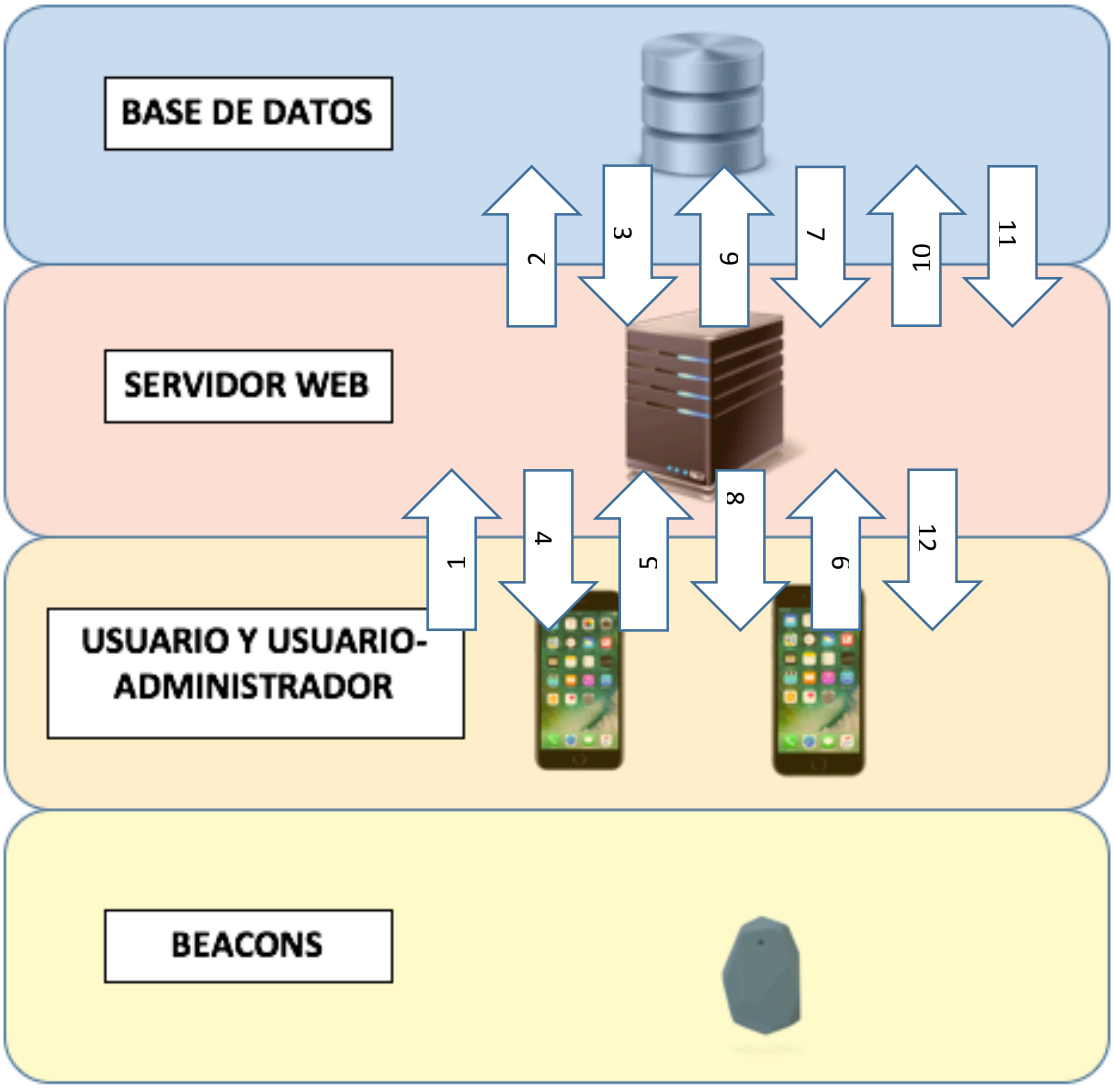


Ilustración 56 - Ejemplo de uso nuevo usuario

La imagen anterior muestra lo que ha sucedido hasta ahora. A continuación, se describe cada uno de los puntos:

1. Fernando se autentica. Rellena el formulario de acceso con el mail y la contraseña de administrador que han puesto a su disposición para realizar las funciones de su puesto, llegando esta información al servidor.
2. El servidor, tras recibir esta información, manda una consulta a la base de datos para localizar el usuario cuyo mail es el introducido por Fernando.
3. Tras realizar la búsqueda, la base de datos le devuelve un JSON con la información relativa al usuario que ha encontrado.
4. El servidor API-Rest, tras hacer una comparación encriptada de la contraseña almacenada y la introducida, manda un mensaje de OK al dispositivo móvil de Fernando, que le permitirá el acceso a la pantalla principal de la aplicación.
5. Como es la primera vez que usa la aplicación, la lista de obras dadas de alta está vacía, por lo que no aparece ninguna. Por tanto, se dispone a dar de alta la primera. Para ello, pulsa el botón 'Nuevo Beacon' y rellena todo el formulario, enviando esta información al servidor.
6. El servidor, tras recibir estos datos, los trata y comprueba si coinciden con el modelo de obra que debería tener. Si esto se cumple, creará un JSON con toda esta información y lo pasará a la base de datos.
7. Si todo ha ido bien, la base de datos devolverá un mensaje satisfactorio.
8. A su vez, este mensaje satisfactorio será enviado al Smartphone de Fernando.
9. Aunque parece que todo ha ido bien, Fernando quiere asegurarse que el sistema ha dado de alta esta primera obra correctamente. Para ello vuelve a la pantalla inicial y recarga la página deslizando su dedo hacia abajo sobre la pantalla. En ese momento, se envía una petición tipo GET al servidor para devolver todos los cuadros (api/cuadro).
10. El servidor, recibe esta petición, y hace consultas a la base de datos para conseguir todos los cuadros que hay actualmente dados de alta en la base de datos.
11. La base de datos mandará todas las obras almacenadas en formato JSON al servidor. Actualmente, solo está dada de alta la única obra que Fernando ha registrado.
12. El servidor, enviará estos datos al Smartphone, que se encargará de, en cada caso, coger la información enviada para que Fernando pueda visualizarla en su aplicación. Así, aparecerá esta primera obra dada de alta.

Tras esto, Amparo ha llegado al museo con la intención de conocerlo en profundidad. Normalmente coge una audioguía porque le gusta enterarse, más allá de la visualización, de la historia y el trasfondo que hay en cada obra. Se da cuenta que hay una nueva aplicación gratuita para visitar el museo, y como es una persona que usa mucho su iPhone, no le importa durante la visita, utilizarlo, poder interactuar con el museo, y, además lo demanda. El trato personalizado dentro de la visita termina por decarntar la balanza y decide descargarse la aplicación en vez de coger la audioguía.

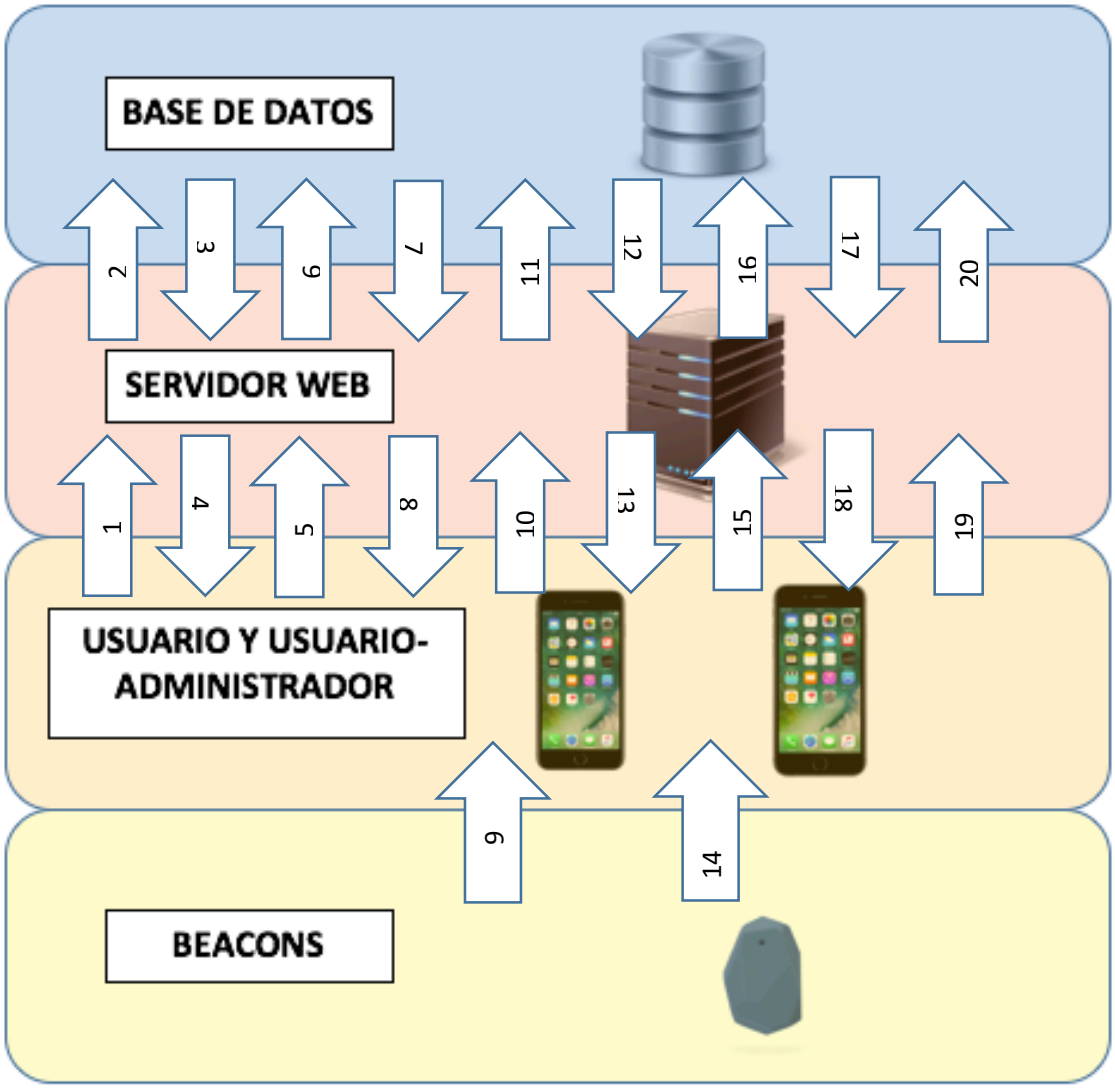


Ilustración 57 - Ejemplo de uso nuevo visitante

1. Amparo se da cuenta que no tiene cuenta en el sistema. Por tanto, elige el botón de registrarse y rellena el formulario de registro, y envía los datos.
2. El servidor, recibe esta información y compara si los datos introducidos coinciden con el modelo de usuario. Como han sido introducidos de manera adecuada, encripta la contraseña y envía esta información en formato JSON a la base de datos, para ser almacenado.
3. La base de datos recibe este JSON y lo almacena, enviando al servidor un mensaje satisfactorio.

4. Puesto que el mensaje ha sido satisfactorio por parte de la base de datos, lo reenvía al Smartphone de Amparo. Así, Amparo se puede dirigir a la pantalla de Login para autenticarse.
5. Amparo, una vez registrada, rellena el formulario de autenticación y pulsa el botón de Login. En ese momento, se envían estos datos al servidor.
6. El servidor pide a la base de datos que le busque el usuario cuyo mail coincide con el introducido por Amparo.
7. La base de datos le devuelve los datos del usuario almacenado en la base de datos en formato JSON.
8. El servidor, comprueba que estos datos coinciden con los introducidos, sin desencriptar la contraseña almacenada. Como coinciden todos los datos introducidos con los almacenados, el servidor manda mensaje satisfactorio al Smartphone y Amparo está ahora autenticada.
9. Una vez autenticada, Amparo se acerca a la primera obra del museo y su Smartphone recibe la primera señal del beacon que tiene más cercano.
10. El Smartphone de Amparo, extrae de la señal emitida por el beacon el minor number, y lo envía al servidor.
11. El servidor recibe este número, y envía una petición a la base de datos para que le devuelva los datos que la obra que está vinculada a ese beacon.
12. La base de datos le devuelve al servidor, en formato JSON, la información solicitada.
13. A su vez, el servidor envía este JSON al Smartphone, que se encargará de mostrarlo en la interfaz de la aplicación que Amparo está utilizando.
14. Esta obra le ha gustado a Amparo, pero decide seguir hasta la siguiente, recibiendo su Smartphone la señal de un nuevo beacon.
15. La aplicación de usuario se da cuenta del cambio de beacon, por lo que envía este cambio al servidor, así como el nuevo beacon que tiene más cercano.
16. El servidor, en primer lugar, envía una solicitud de actualización a la base de datos del JSON de la primera obra que ha visto Amparo, incrementando en una unidad el contador de transición a la nueva obra que está visualizando Amparo. Además, solicita la información de la nueva obra.
17. La base de datos devuelve en formato JSON la información de la obra que actualmente está visualizando Amparo.
18. A su vez, el servidor envía esta información a la aplicación, que muestra los datos de la nueva obra al usuario.
19. Esta obra le ha encantado a Amparo. Ha estado bastante tiempo visualizándola y la quiere evaluar con un 10, y pulsa el botón 'Puntuar Obra'. La aplicación, envía la información del beacon que ha evaluado, el tiempo que ha permanecido observándolo, y la puntuación.
20. El servidor recoge esta información y envía a la base de datos una petición de actualización del usuario de Amparo en base de datos, actualizando esta nueva información.

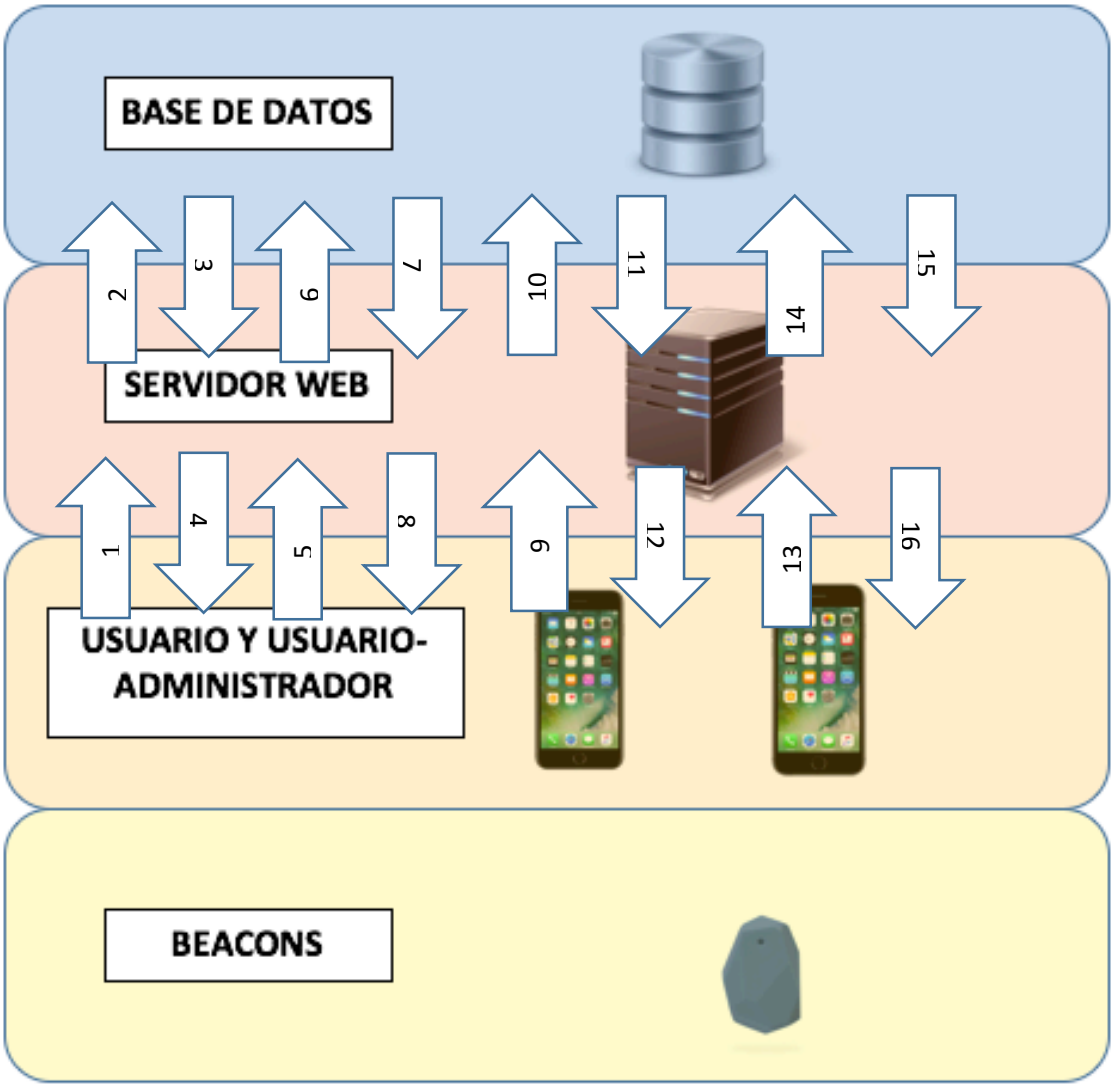


Ilustración 58 - Ejemplo de uso nuevo usuario (II)

Fernando, decide consultar como está funcionando el sistema y como están interactuando los visitantes con la aplicación y el museo.

1. Fernando quiere ver la información de una de las obras. Por tanto, en su pantalla principal, pulsa sobre ella. La aplicación, que tiene la información de todas las obras por realizar la consulta para la visualización de la pantalla inicial, y para evitar hacer más peticiones de las necesarias al servidor, carga la información de la obra sobre la que ha clickado Fernando en una nueva pantalla. En ese momento, Fernando se da cuenta que una de las obras tiene un error en la fecha, por lo que selecciona el botón de actualizar y escribe el valor que quiere cambiar en el formulario y pincha en confirmar. En ese momento, se envía al servidor, desde la aplicación, el minor number de la obra que hay que actualizar y los datos.
2. El servidor recibe esta petición y realiza una orden de actualización a la base de datos de la información enviada en la obra almacenada cuyo minor number coincide con el enviado.

3. La base de datos devuelve un mensaje de OK.
4. Que a su vez el servidor, envía a la aplicación.
5. Ahora Fernando, con la información de la obra correcta, decide ver las estadísticas de esta. Por tanto, pincha en 'Estadísticas'. Puesto que estos datos si pueden estar cambiando en tiempo real por la interacción de los usuarios con el sistema, ahora si se realiza una petición al servidor. La aplicación enviará el minor number del beacon del que quiere saber los contadores de interacciones, el sumatorio del tiempo de visualización, y la cantidad de visualizaciones que ha tenido esa obra.
6. El servidor, realiza una consulta para obtener la información de la obra vinculada al minor number de ese beacon.
7. La base de datos, devuelve un JSON con la información completa de esta obra.
8. A su vez, el servidor la reenvía a la aplicación. Esta, realiza los cálculos de visualización media del total de usuarios, dividiendo el sumatorio del tiempo de visualización entre el tiempo de visualizaciones. A su vez, genera una gráfica con el resto de beacons al que se ha pasado desde el seleccionado actualmente.
9. Al observar la gráfica, Fernando se da cuenta que hay uno de las obras no está siendo visualizada por el lugar donde está colocada, y tiene muy poco tiempo medio de visualización. Fernando habla con sus superiores y les sugiere que la obra que debe ir colocada en su lugar debe ser una que tenga un mayor tiempo de visualización para aprovechar mejor todas las partes del museo, y reubicar esta. Al obtener la autorización, decide borrar esta obra del sistema para poder dar de alta la nueva. Por ello, la selecciona en su aplicación y posteriormente elige el botón 'Borrar'. En ese momento, la aplicación envía dicha solicitud al servidor con el minor number del beacon vinculada a la obra que se quiere reubicar y quitar del sistema.
10. El servidor recoge esta información y manda la orden a la base de datos de borrar del sistema el registro cuyo minor number coincide con el enviado.
11. La base de dato lo borra y envía un mensaje de OK.
12. Y a su vez, el servidor lo reenvía a la aplicación.
13. Fernando, antes de efectuar el cambio, quiere asegurarse que ha sido borrado, por lo que vuelve a la pantalla principal de la aplicación y desliza el dedo hacia abajo para recargar la página. En ese momento, la aplicación envía la solicitud de información de todos los cuadros almacenados en el sistema.
14. El servidor recoge esta solicitud y la envía a su vez a la base de datos.
15. Esta, devuelve un JSON con la información de todas las obras almacenadas, enviándolas al servidor.
16. El servidor a su vez lo envía a la aplicación, que muestra en una tabla todas las obras. Ahora Fernando puede ver que se ha borrado correctamente al no estar en la lista, y se dispone, ahora sí, a realizar el cambio de ubicación.

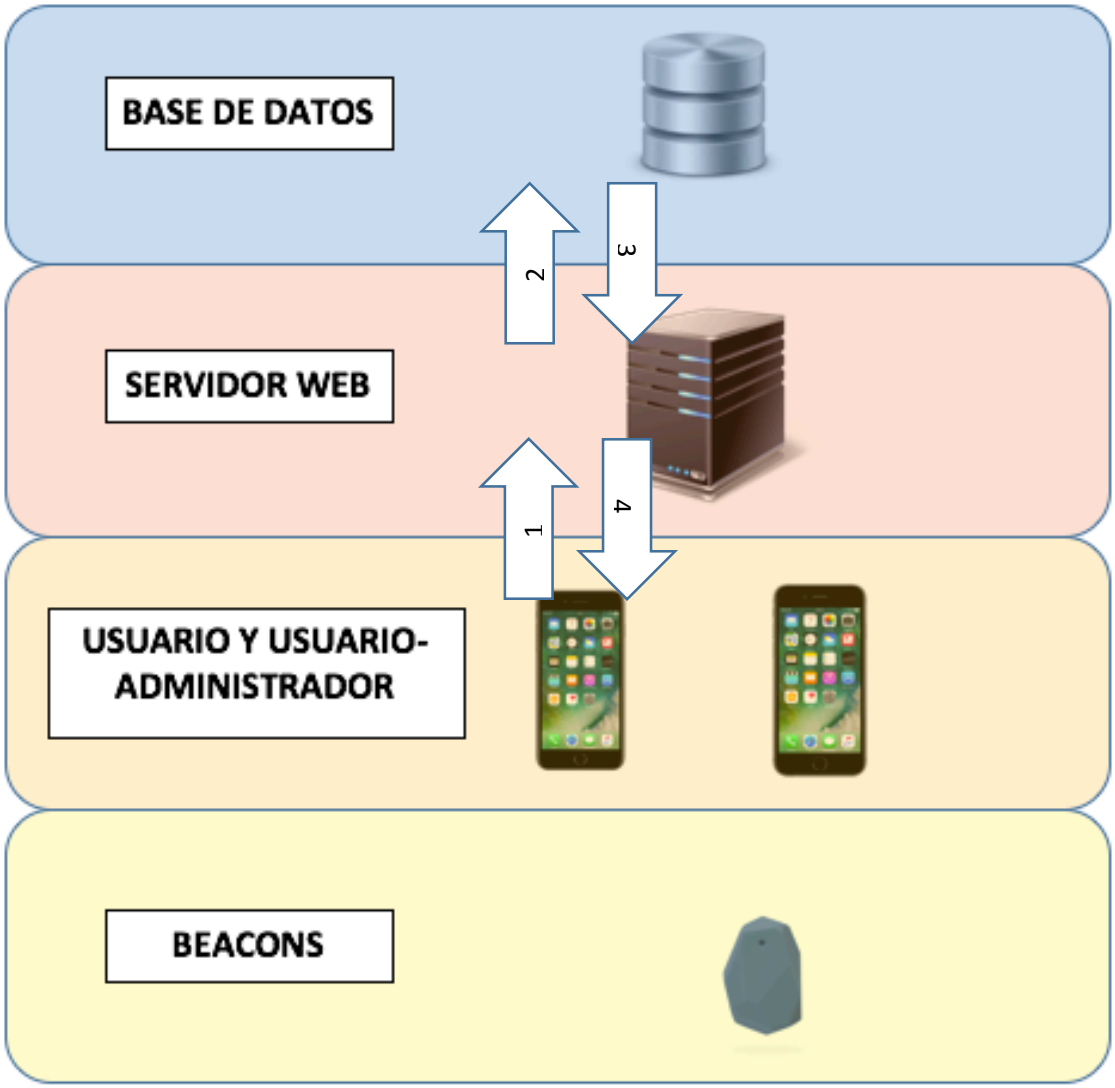


Ilustración 59 - Ejemplo de uso cierre sesión visitante

1. Amparo, esta vez, ha decidido que, por el momento, ya está bien, y quiere irse del museo. Por tanto, presiona el botón de cerrar sesión. Puesto que ha evaluado una obra y ha estado visualizando durante largo tiempo, existe una regla de negocio que, cumpliendo una valoración X por un tiempo de visualización Y, y siendo mayor que Z, se dará un cupón de un 25% de descuento de artículos de esa obra en la tienda de souvenirs. Por esto, se envía una solicitud al servidor del cupón vinculado al beacon mejor valorado.
2. El servidor realiza la petición de los datos de la obra cuyo minor number del beacon coincide con el enviado.
3. La base de datos devuelve, en formato JSON, esta información.
4. A su vez, es enviado a la aplicación, que recoge el atributo de la URL del cupón de esta obra y la muestra al usuario para que pueda mostrarlo en la tienda de souvenirs.

3.6. PLANIFICACIÓN Y PRESUPUESTO

3.6.1. PLANIFICACIÓN

En el presente apartado se describe la **planificación** que ha sido necesaria para el **desarrollo del sistema**. Estas fases son **análisis del sistema, diseño del sistema, implementación y** por último **pruebas**. Se debe tener en cuenta que estas fases no son secuenciales, y que conforme se van desarrollando unas, otras pueden sufrir modificaciones y revisiones, por lo que algunas fases se solapan en el tiempo.

ANÁLISIS DEL SISTEMA

Permite **definir el problema**, es decir, obtener las especificaciones sobre qué debe hacer el sistema, sin entrar en cómo debería hacerlo. En esta fase se extraerán las funcionalidades de las que constará el sistema. Por tanto, si durante el desarrollo de la solución y durante la implementación surgieran nuevas funcionalidades, se incluirán en esta parte del análisis del sistema, para seguir luego con su diseño e implementación.

DISEÑO DEL SISTEMA

Consiste en el **modo en que se resuelve el problema** analizado, descrito e identificado en el apartado anterior. Además de explicar cómo se solucionará el problema, se identificarán las tecnologías que se utilizarán para ello, apoyado en una arquitectura diseñada para el sistema. Si durante la fase de implementación se identificaran nuevas formas de solucionar problemas, o surgieran nuevas funcionalidades, se añadirían a este apartado de diseño.

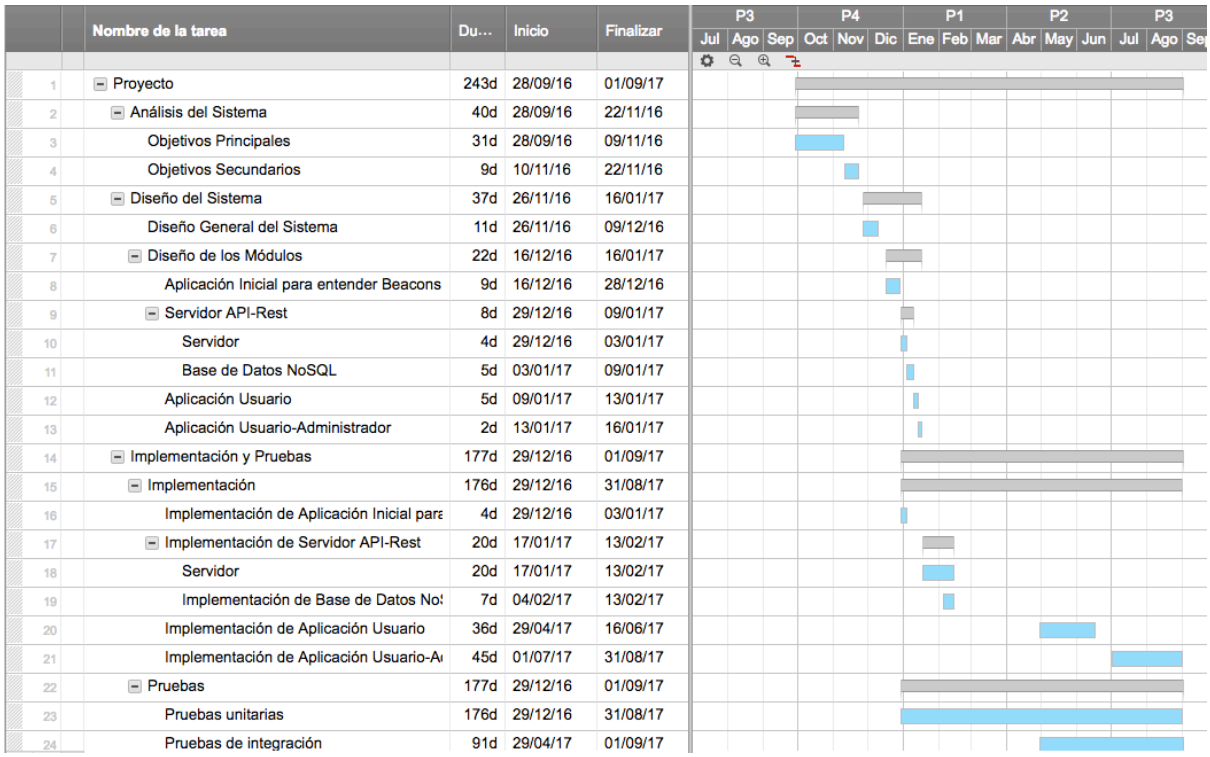
IMPLEMENTACIÓN Y PRUEBAS

Se procederá al **desarrollo y ejecución de la solución** de los dos apartados anteriores, utilizando las tecnologías y herramientas mencionadas en el diseño del sistema. Se entiende que la fase de implementación y pruebas tampoco es secuencial, y que durante una y otra se puede ocasionar nuevas acciones en la contraria y viceversa. Además, al tratarse de una solución que tiene varios subsistemas, necesitamos el funcionamiento de cada uno por separado y el funcionamiento a la vez. Por ello, se implementará cada uno y se probará por separado y posteriormente en conjunto.

DIAGRAMA DE GANTT

Una de las formas mejor y más eficiente de **mostrar visualmente** la planificación de un proyecto, es mediante un Diagrama de Gantt. En estos diagramas, se muestra como se prolongan las distintas tareas

en el tiempo, a la vez que se puede ver cómo interactúan y solapan unas fases con otras y el orden en que se llevan a cabo a lo largo del tiempo.



	Nombre de la tarea	Du...	Inicio	Finalizar
1	Proyecto	243d	28/09/16	01/09/17
2	Análisis del Sistema	40d	28/09/16	22/11/16
3	Objetivos Principales	31d	28/09/16	09/11/16
4	Objetivos Secundarios	9d	10/11/16	22/11/16
5	Diseño del Sistema	37d	26/11/16	16/01/17
6	Diseño General del Sistema	11d	26/11/16	09/12/16
7	Diseño de los Módulos	22d	16/12/16	16/01/17
8	Aplicación Inicial para entender Beacons	9d	16/12/16	28/12/16
9	Servidor API-Rest	8d	29/12/16	09/01/17
10	Servidor	4d	29/12/16	03/01/17
11	Base de Datos NoSQL	5d	03/01/17	09/01/17
12	Aplicación Usuario	5d	09/01/17	13/01/17
13	Aplicación Usuario-Administrador	2d	13/01/17	16/01/17
14	Implementación y Pruebas	177d	29/12/16	01/09/17
15	Implementación	176d	29/12/16	31/08/17
16	Implementación de Aplicación Inicial para	4d	29/12/16	03/01/17
17	Implementación de Servidor API-Rest	20d	17/01/17	13/02/17
18	Servidor	20d	17/01/17	13/02/17
19	Implementación de Base de Datos No!	7d	04/02/17	13/02/17
20	Implementación de Aplicación Usuario	36d	29/04/17	16/06/17
21	Implementación de Aplicación Usuario-A	45d	01/07/17	31/08/17
22	Pruebas	177d	29/12/16	01/09/17
23	Pruebas unitarias	176d	29/12/16	31/08/17
24	Pruebas de integración	91d	29/04/17	01/09/17

Ilustración 60 - Diagrama de Gantt

3.6.2. PRESUPUESTO

En el presente apartado se detallarán los **costes del desarrollo** del proyecto. Comprenden costes de hardware, software y de personal.

HARDWARE

Como se ha comentado en el apartado 1.3 de este documento, para el desarrollo de la solución se ha empleado un ordenador personal, con las características detalladas a continuación:

Marca	MacBook Pro (Retina 13 pulgadas)
Procesador	2,7 GHz Intel Core i5
Memoria Ram	8GB 1867 MHz DDR3
Disco Duro	250 GB

Tabla 23 - Coste de computadora en el Proyecto

Este equipo tiene un coste aproximado de 1.500 €. Suponiendo una **amortización lineal** a cinco años, y al haber sido usado en el proyecto durante 11 meses, se le imputa un coste de 275 €.

En cuanto a los dispositivos móviles utilizados para las pruebas y desarrollo de la aplicación, recordemos las características que tenían:

Marca	iPhone 6s 16GB 4,7”
Procesador	Chip A9 (coprocesador M9) con arquitectura de 64 bits
Memoria Ram	2 GB
Disco Duro	16 GB
Sistema Operativo	iOS 10

Tabla 24 - Coste de smartphone en proyecto (I)

Marca	**iPhone 7 Plus 128GB 5,5”**
Procesador	Chip A10 (coprocesador M10) con arquitectura de 64 bits
Memoria Ram	3 GB
Disco Duro	128 GB
Sistema Operativo	iOS 10

Tabla 25 - Coste de smartphone en proyecto (II)

Para estos dos dispositivos, suponiendo una **amortización para este hardware de 3 años**, y al haber sido utilizado durante un total de 11 meses, teniendo un coste de 1.100 euros para el primer dispositivo y 550 para el segundo, se imputa un coste de 336,11 € y 168,05 €.

Marca	Estimote
Procesador	32-bit ARM Cortex MO CPU
Memoria Ram	256 KB

Tabla 26 - Coste Beacons en el proyecto

En el caso de los beacons, se estima un **periodo de amortización de 2 años** y han tenido un coste de 178 euros. Por tanto, se imputa un coste de 81,58 €.

PERSONAL

Al ser un Trabajo de Fin de Grado, el trabajo ha sido desempeñado **únicamente por una persona**. No obstante, con la intención de realizar el estudio de presupuesto como si se hubiese realizado en un entorno corporativo, se ha desglosado diferentes puestos en función del trabajo desempeñado, siendo **analista, diseñador, programador y encargado de pruebas**.

Puesto	Horas de trabajo	Salario por Hora	Coste total
Analista	192 horas	20	3.840 €
Diseñador	178 horas	18	3.204 €
Programador	422 horas	16	6.752 €
Encargado de pruebas	100 horas	15	1.500 €
Total	**892 horas**		**15.296 €**

Tabla 27 - Coste de personal en el proyecto

SOFTWARE

En el caso del software utilizado, el precio del sistema operativo y el software para el desarrollo de la aplicación móvil es **gratuito** y va incluido al adquirir el hardware. El resto de software utilizado para el desarrollo de la base de datos y del servidor es gratuito y no ha tenido ningún tipo de coste. La licencia de desarrollo de aplicaciones iOS también es gratuita para desarrollar, y no tiene coste a no ser que se publique la aplicación en el Apple Store.

RESUMEN

A continuación, se muestra un cuadro resumen de los costes estimados anteriormente. Se debe tener en cuenta que se ha incluido un **coste extra** por problemas que puedan surgir en el proyecto, como puede ser una reorganización del proyecto, que se alargue más de lo planificado, etc. equivalente al 10% extra de los costes totales.

Elemento	Coste
Hardware	860,74 €
Software	0 €
Personal	15.296 €
Extra 10%	1.615,68 €
Total	**17.772,42 €**

Tabla 28 - Resumen costes proyecto

3.7. MANUAL DE USUARIO

En este apartado se explicará **como un usuario debe interactuar con el sistema y las capacidades que tiene el mismo**, para que así pueda usarlo de la mejor manera posible. Para ello se usarán capturas de cada pantalla que tiene la aplicación para así poder mostrar al usuario cuales son las opciones que tiene. El manual se dividirá en dos partes, una por cada aplicación, pudiendo ser como se ha mencionado a lo largo del trabajo, usuario o usuario-administrador.

3.7.1. APLICACIÓN DE USUARIO

En primer lugar, se debe tener en cuenta que en la aplicación hay una parte **pública**, donde cualquier usuario que se descargue la aplicación puede acceder, y una parte **privada**, a la que se accede mediante autenticación. A continuación, se enumeran las distintas acciones que puede realizar el usuario:

REGISTRO

La primera acción que debe hacer un usuario que accede a la aplicación por primera vez es **registrarse**. Para ello, según ejecute la aplicación, aparecerá la pantalla de inicio, que es pública, y a partir de la cual puede acceder a la pantalla de registro, tal y como vemos en la ilustración que a continuación se muestra a la izquierda. Pinchando sobre el botón 'Regístrate', llegaremos a la pantalla mencionada, como se muestra abajo a la derecha.

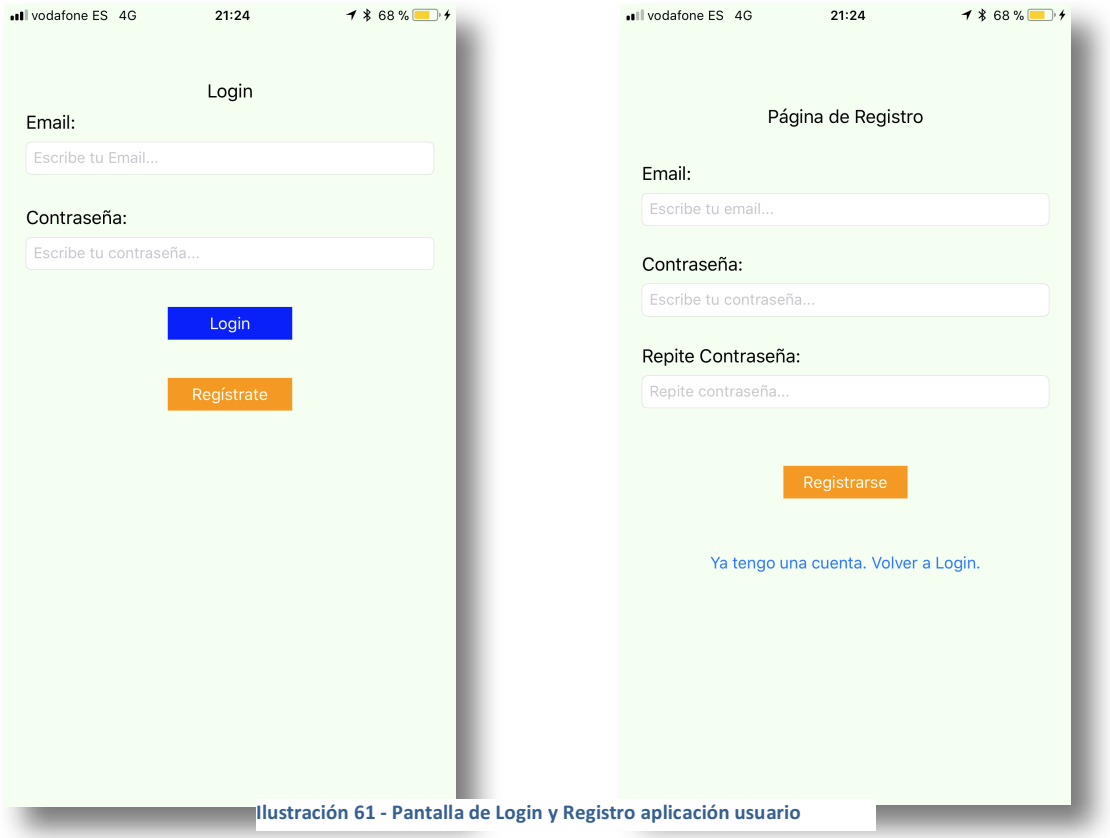


Ilustración 61 - Pantalla de Login y Registro aplicación usuario

Una vez llega el usuario a la pantalla de registro, debe rellenar el formulario que aparece, escribiendo su dirección de email y la contraseña. Tras esto, el usuario deberá pinchar en 'Registrarse' y será redirigido a la pantalla de **Login**, previo mensaje satisfactorio del registro.

Si el usuario accediese a esta página cuando ya estaba registrado, podrá volver a la pantalla de Login pinchando en 'Ya tengo una cuenta. Volver a Login.'.

AUTENTICACIÓN

Una vez el usuario visualice la pantalla de Login, ya sea tras el registro, o cada vez que acceda a la aplicación mediante una sesión nueva, deberá rellenar el mail y contraseña con la que se registró. Si todo va bien y no muestra ningún mensaje de error, el usuario sabrá que todo ha ido bien, tras pinchar en el botón 'Login', si es redirigido directamente a la pantalla para **usuarios autenticados**.

USO DE LA APLICACIÓN EN EL INTERIOR DEL MUSEO

Cuando el usuario se haya **autenticado satisfactoriamente**, para poder usar la aplicación únicamente tendrá que andar por el museo y acercarse a la obra de la que quiera obtener más información. Cuando se acerque a una nueva obra, su pantalla cambiará automáticamente y mostrará información relevante de dicha obra.

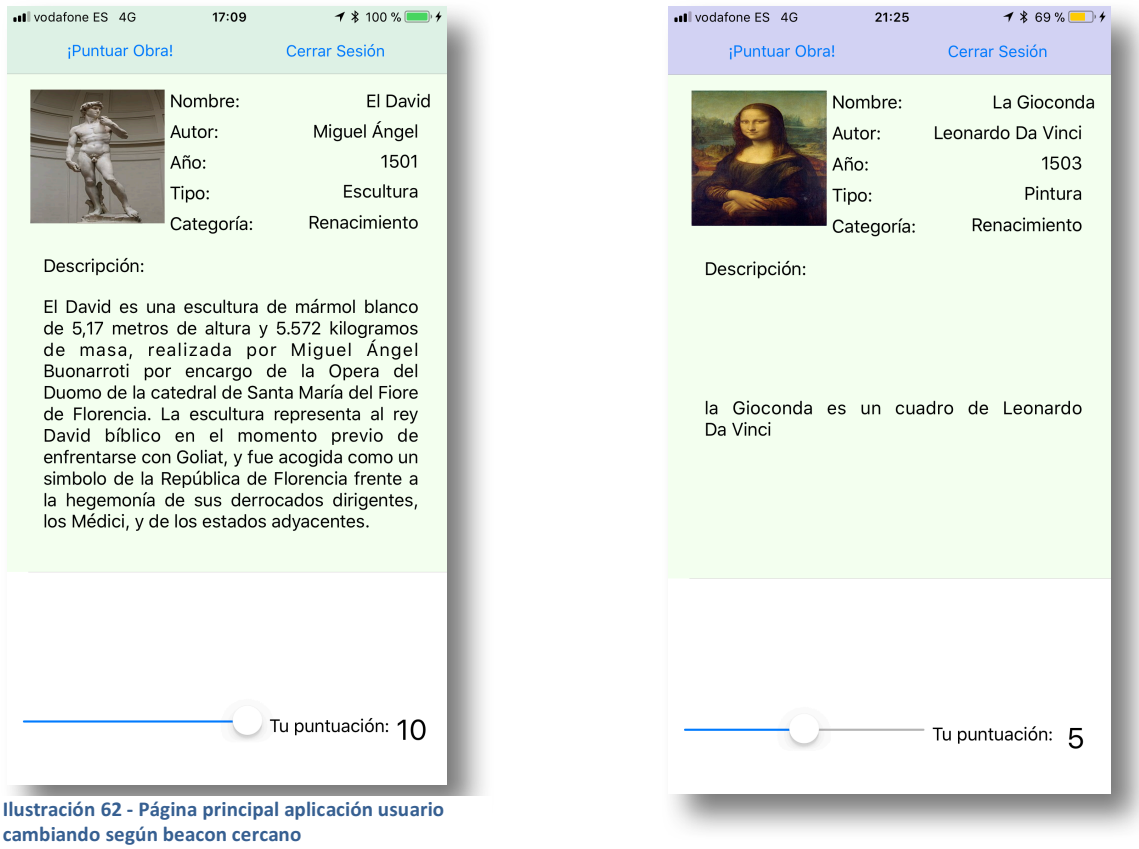


Ilustración 62 - Página principal aplicación usuario cambiando según beacon cercano

VALORACIÓN DE OBRA

En la pantalla principal una vez se ha autenticado el usuario, puede observar en la parte inferior un botón deslizable para cambiar la que sería la puntuación que le da a la obra que tiene más cercana y de la que está consultando información. Una vez ha deslizado el botón hasta **obtener la valoración** que quiere dar, basta con presionar el botón '¡Puntuar Obra!' situada en la parte superior izquierda de la pantalla. Si todo ha sucedido correctamente, aparecerá un mensaje agradeciendo la votación.

CERRAR SESIÓN

También en la pantalla principal, una vez el usuario haya finalizado su recorrido en el museo y desee finalizar la visita y cerrar la aplicación, dispone de un botón para ello en la esquina superior derecha. Le recomendamos que el cierre de sesión lo haga al finalizar el recorrido y justo antes de pasar por la tienda de souvenirs, ya que el museo puede tener en ese momento algún gesto comercial con usted en dicho establecimiento.

3.7.2. APLICACIÓN DE USUARIO ADMINISTRADOR

Al igual que la aplicación de usuario, dispone de una parte pública y una privada. A diferencia con la aplicación anterior, **no hay forma de registrarse** en esta aplicación, pudiendo hacerlo solo el administrador del sistema, otorgando al administrador de la solución del museo un usuario y una clave determinada.

AUTENTICACIÓN

Para poder autenticarse, el usuario deberá introducir el usuario y contraseña facilitada por el administrador del sistema y que permitirá el acceso a la aplicación, rellenando los campos mostrados y clicando sobre el botón 'Login'.

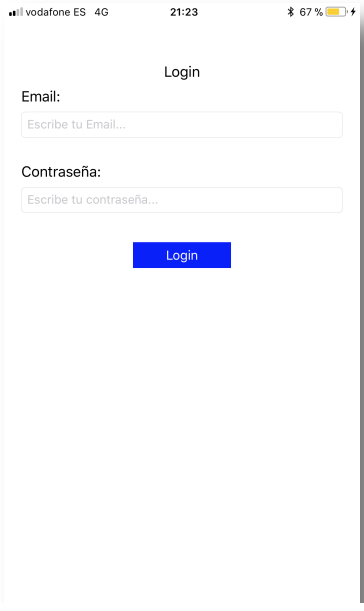


Ilustración 63 - Pantalla login aplicación usuario-administrador

CONSULTAR OBRAS DADAS DE ALTA EN EL SISTEMA

Una vez autenticado el usuario-administrador, en la pantalla principal aparecerá un **listado de las obras** que han sido dadas de alta en el sistema, pudiendo hacer clic sobre cada una de ellas.

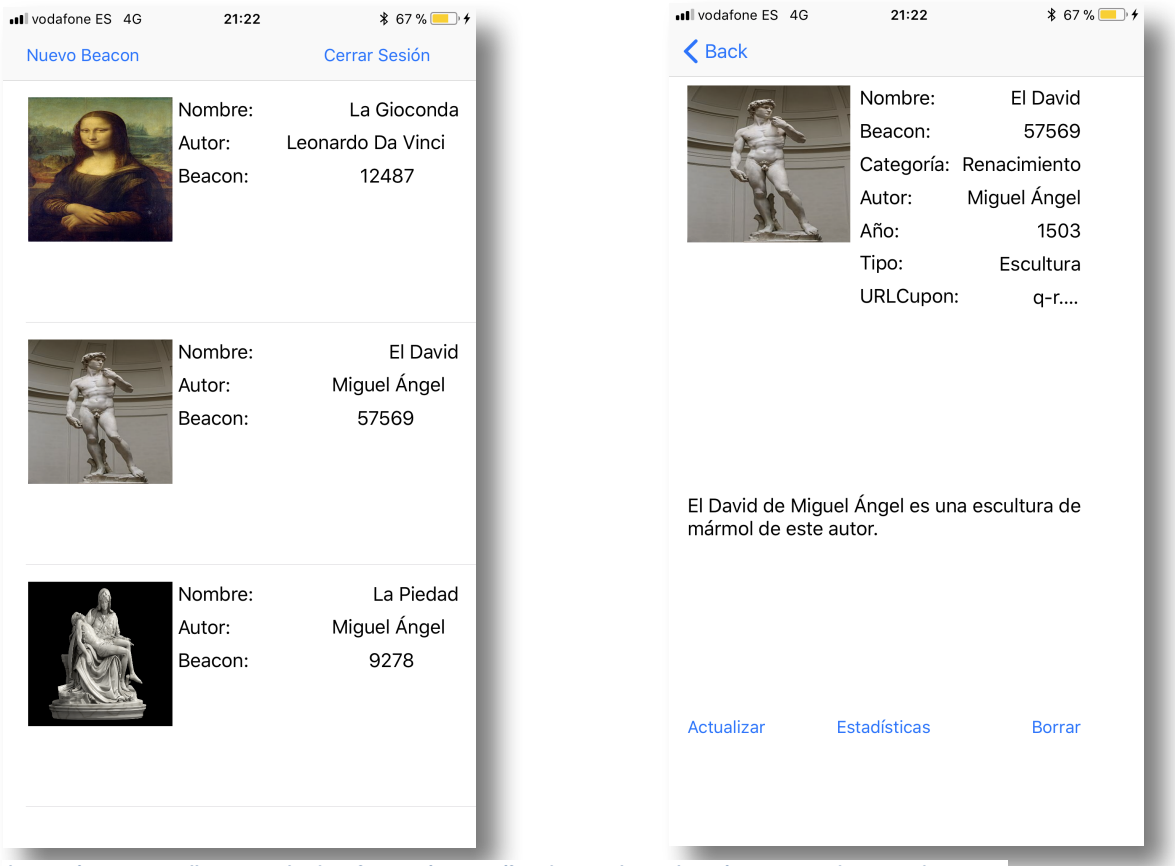


Ilustración 64 - Pantalla principal y de información específica de una obra aplicación usuario-administrador

CONSULTAR INFORMACIÓN AMPLIADA DE UNA OBRA DADA DE ALTA

En la pantalla principal, el listado que aparece permite seleccionar una obra para acceder a **información más ampliada** de la misma, y que es útil para el administrador de la solución en el museo.

Además, podrá acceder a la actualización de los valores mostrados si fuese necesario, consultar sus **estadísticas**, o **borrar** una obra del sistema, no apareciendo en el listado cuando recargáramos la pantalla principal y no estando visible para los visitantes del museo. Podemos observar dicha pantalla en la imagen superior derecha.

ACTUALIZAR OBRA

Si el administrador de la solución en el museo detectara que hay algún error en la información que se muestra de una obra al visitante, puede actualizarla seleccionando esa obra en el listado general, y haciendo clic en el botón 'Actualizar'.

Esta acción redirigirá al usuario a un formulario en el que se muestra el beacon que está vinculado a la obra seleccionada, y nos permite escribir sobre el campo que deseamos actualizar o cambiar su información. En este caso no es necesario escribir todos los campos, sino únicamente **el que el usuario quiera actualizar**, manteniéndose los demás con los mismos datos, y seleccionar el botón 'Confirmar'. Si finalmente no desea actualizar ninguna información, puede volver a la pantalla anterior pinchando en el botón 'Back' de la parte superior izquierda de su pantalla.

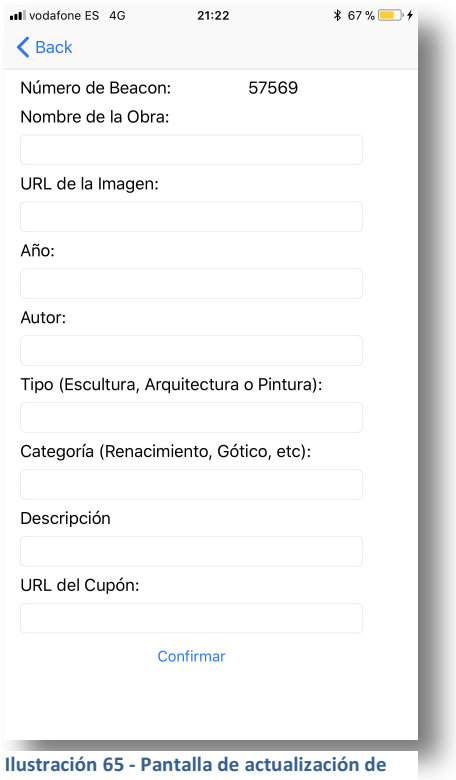


Ilustración 65 - Pantalla de actualización de información beacon

ACCEDER A ESTADÍSTICAS DE UNA OBRA

De la misma forma que accedemos a la actualización de una obra, el usuario puede seleccionar la opción '**Estadísticas'** en esa misma pantalla.

De esta forma, se mostrará la información estadística relativa a la obra seleccionada, pudiendo ver el **tiempo medio** que un usuario permanece observando la obra, y cuál es el **número de veces que se pasa**

desde la obra seleccionada a las demás. Una vez terminada la visualización, puede volver atrás presionando el botón ‘Back’ en la esquina superior izquierda.

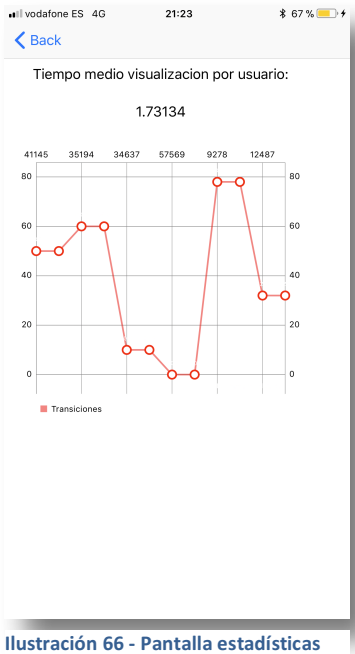


Ilustración 66 - Pantalla estadísticas beacon

VINCULAR NUEVO BEACON A UNA OBRA

Si lo que el usuario-administrador de la solución en el museo desea es dar de alta una nueva obra en el sistema, en la pantalla principal, donde aparece el listado de obras dadas de alta actualmente, aparece el botón ‘Nuevo Beacon’ en la esquina superior izquierda. Esto nos conducirá a un formulario en el que todos los campos son obligatorios. Una vez rellenados todos los campos, basta con seleccionar el botón ‘Confirmar’ para dar de alta un nuevo beacon vinculado a una obra determinada, y desde ese instante estará listo para ser visualizado por los visitantes del museo.

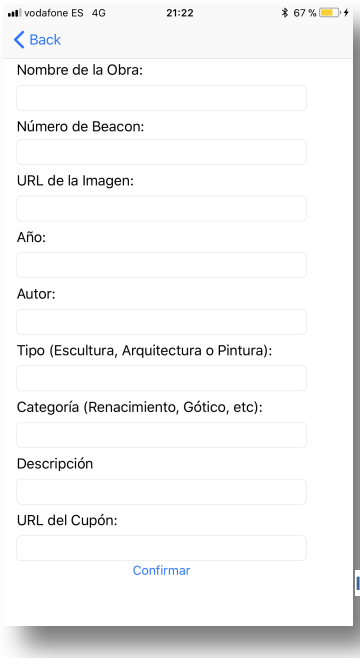


Ilustración 67 - Pantalla nuevo beacon

CERRAR SESIÓN

En la pantalla principal, en el momento en que el usuario-administrador de por finalizada su labor, puede cerrar sesión pinchando sobre el botón de la esquina superior derecha que describe esta misma acción.

3.8. MARCO REGULATORIO

En cuanto al aspecto legal de nuestra solución, la única ley relevante en cuanto a este proyecto de software es la Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal o **LOPD**.

Esta ley busca proteger la seguridad de ciertos datos personales de los clientes. En el caso de nuestra solución, como se indicó en los requisitos de seguridad, las contraseñas siempre son enviadas empleando un algoritmo criptográfico, y almacenadas en base de datos codificadas. También, ante la comprobación de las contraseñas durante la autenticación, no se decodifica la contraseña almacenada, si no que se hace una **comparativa de encriptación**.

Ninguno de los datos de los usuarios solicitados será empleados para ningún otro asunto ni será difundido, salvo requerimiento por parte de autoridad competente con orden judicial.

4. CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se analizará los objetivos alcanzados con respecto a los establecidos al comienzo del documento, y se enumerarán posibles mejoras que se podrían realizar en el futuro.

4.1. CONCLUSIONES

4.1.1. GENERAL

En lo personal, puedo afirmar que ha sido un proyecto exigente y completo. Desde el planteamiento inicial quise que el proyecto agrupara varios aspectos puestos en práctica durante la carrera además de usar algunos productos que por la experiencia durante mis prácticas de empresa en Oracle podrían ser beneficiosas saber por la tendencia que tiene el mercado en la actualidad y por enriquecer mi currículo y el proyecto en sí. Ha sido un proyecto con distintas fases y productos usados en la solución muy distintos, que ha sido todo un reto de desarrollo por separado e integración en la parte final.

Todo esto, unido a que mientras realizaba el trabajo de fin de grado, y como casi toda la carrera, lo he compaginado con mi trabajo lo que ha supuesto un reto organizativo aún mayor. Por todo ello, estoy satisfecho por el resultado final y creo que es una buena forma de acabar mi Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas.

4.1.2. SOLUCIÓN

Al comienzo de este documento se establecieron una serie de objetivos que debía cumplir la solución, y debía ser de una forma determinada y con unos tipos de producto determinado que enriquecieran el conjunto completo del proyecto.

Uno de los objetivos establecidos fue el usar tecnologías que no fueran las habituales su uso a lo largo de la carrera, pero que están muy presentes en el mercado, y así complementar lo aprendido en la universidad con otros conocimientos que en el mercado laboral pueden ser de utilidad.

Paradigmas como el IOT, la digitalización de espacios, el uso de bases de datos NoSQL muy presentes hoy en día en paradigmas de Big Data, el uso de aplicaciones móviles, y la integración de todos ellos, eran requisitos fundamentales que puedo decir se han cumplido perfectamente. Digitalizando espacios con beacons, dando a las obras acceso a internet (Internet de las Cosas), dar al usuario un trato personalizado y que su experiencia sea interactiva, como reclaman los usuarios en la actualidad. Ayudar a las empresas, en este caso al museo, a extraer valor del comportamiento de sus visitantes, maximizar

las ventas, permitir fidelizar a los usuarios… todo ello eran requisitos que, más allá de las tecnologías a utilizar, eran parte importante de la solución y también se ha cumplido con éxito.

4.2. TRABAJOS FUTUROS

En cuanto a posibles mejoras que se podrían realizar en la solución en el futuro, en primer lugar y la más obvia, es ampliar la aplicación de usuario tambien a un sistema operativo Android. Es algo relativamente sencillo pero que en este proyecto me decanté por sistema operativo iOS por, entre otras cosas, el no haberlo usado en la universidad, mientras que Android sí.

Ha sido un trabajo que ha sido muy exigente en cuanto a funcionalidad. Por tanto, aunque no ha sido mala, la interfaz de usuario se puede mejorar más y hacerla aún más atractiva.

En cuanto a la base de datos, hay sistemas Big Data, que aunque no es nuestro caso, usan distintos tipos de base de datos en función de las capacidades que se requiera de cada uno y de los resultados que se esperen de ellos. Una mejora importante sería combinar una base de datos NoSQL con una SQL, que tenga datos que necesiten de un relacional como podrían ser la base de datos de productos de la tienda de souvenir, y poder cruzarlo con los cupones que utilicen los usuarios, mejorando así decisiones de ventas.

En cuanto a la inteligencia de negocio, sería interesante implementar una interfaz que permitiera al usuario-administrador establecer según que reglas de negocio para según qué usuarios para que, en función de su comportamiento, reciban unos cupones u otros. Se ha implementado una regla de negocio muy básica, pero sería interesante que pudiera elegirla y manipularla el usuario.

5. BIBLIOGRAFIA

[1] Telefónica, "*Smart Cities: un primer paso hacia la internet de las cosas: Telefónica"*, F. Fundación Telefónica. Informe 2011
URL: https://books.google.es/books?id=wZLmCgAAQBAJ

[2] Cornetta, G., "*Internet de las cosas: la hoja de ruta hacia un mundo conectado en red y sus implicaciones en el sector educativo"*, 2016, CEU Ediciones.
URL: https://books.google.es/books?id=xrY2ngAACAAJ

[3] Silvia Leal, *"e-Renovarse o morir"* , 2015, LID Editorial.
URL: https://proquest-safaribooksonline-com.biblioteca5.uc3m.es/9788483562703

[4] Luis Joyanes Aguilar, *"Big Data, Análisis de grandes volúmenes de datos en organizaciones"*, Alfaomega Grupo Editor, 2016
URL: https://books.google.es/books?id=1GywDAAAQBAJ

[5] Antonio Sarasa, "*Introducción a las bases de datos NoSQL usando MongoDB"*, e-libro, Corp, Editorial UOC, 2016
URL: https://books.google.es/books?id=-AyUAQAACAAJ

[6] BBVAOPEN4U, "*API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos",* BBVA, 2016
URL: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[7] BBVAOPEN4U, "*APIs para el Internet de las Cosas: ThingSpeak, Pachube y Fitbit",* BBVA, 2016
URL: https://bbvaopen4u.com/es/actualidad/apis-para-el-internet-de-las-cosas-thingspeak-pachube-y-fitbit

[8] BBVAOPEN4U, "*Javascript. PostCSS: qué es y por qué es tan popular",* BBVA, 2016
URL: https://bbvaopen4u.com/es/actualidad/postcss-que-es-y-por-que-es-tan-popular

[9] Chodorow, K. and Dirolf, M., "*MongoDB: The Definitive Guide*", O'Reilly Media, 2010
URL: https://books.google.es/books?id=BQS33CxGid4C

[10] Chodorow, K., "*50 Tips and Tricks for MongoDB Developers*", O'Reilly Media, 2011
URL: https://books.google.es/books?id=IZiyEX_2JtoC

[11] Dayley, B., "*Node.js, MongoDB, and AngularJS Web Development*", Developer's Library, Pearson Education, 2014
URL: https://books.google.es/books?id=8kTCAwAAQBAJ

[12] Satheesh, M. and D'mello, B.J. and Krol, J., "Web Development with MongoDB and NodeJS", Packt Publishing, 2015
URL: https://books.google.es/books?id=4QKACwAAQBAJ

[13] Espada, J.P., "*Desarrollo Guiado de Páginas Web Dinámicas con Node. Js Express, MongoDB y Uikit*", CreateSpace Independent Publishing Platform, 2016
URL: https://books.google.es/books?id=K5xMMQAACAAJ

[14] Gonzalo, E.F. and Academy, I.T.C., "*Aprende a Programar Swift: Programación iOS. 2ª Edición*", CreateSpace Independent Publishing Platform, 2016
URL: https://books.google.es/books?id=396MCwAAQBAJ

[15] Gast, M.S., "*Building Applications with IBeacon: Proximity and Location Services with Bluetooth Low Energy*", O'Reilly Media, 2014
URL: https://books.google.es/books?id=JiuiBAAAQBAJ

[16] Gast, M.S., "*Building Applications with IBeacon: Proximity and Location Services with Bluetooth Low Energy*", O'Reilly Media, 2014
URL: https://books.google.es/books?id=JiuiBAAAQBAJ

[17] "*The Internet in Real Time*",
URL: https://www.webpagefx.com/internet-real-time/

[18] Ditrendia, "*Informe Mobile en España y en el Mundo 2016*", Ditrendia: Digital marketing Trends, 2016
URL: http://www.amic.media/media/files/file_352_1050.pdf

[18] Acens, "*Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar*", AcensWhitePapers, 2016
URL: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

6. ANEXOS

6.1. ANEXO I: GLOSARIO DE ACRÓNIMOS Y DEFINICIONES

ACRÓNIMOS

- **API:** Aplication Programming Interface. Interfaz de programación de aplicaciones.
- **CPU:** Central Processing Unit. Unidad Central de Procesamiento.
- **GB:** GigaByte.
- **GPS:** Global Positioning System. Sistema de Posicionamiento Global.
- **SDK:** Software Development Kit. Kit de Desarrollo de Software
- **HTML:** HyperText Markup Language. Lenguaje de Marcas de Hipertexto.
- **Hz:** Hercio.
- **GHz:** Gigahercio.
- **IoT:** Internet of Things. Internet de las Cosas.
- **JSON:** JavaScript Object Notation.
- **LTS:** Long Term Support. Soporte de largo plazo.
- **No-SQL:** Not-Only SQL.
- **RAM:** Random Access Memory. Memoria de acceso aleatorio.
- **SQL:** Structured Query Language.

DEFINICIONES

- **Back-End:** Parte del software que procesa la entrada desde el front-end.

- **Big Data:** Paradigma que hace referencia a un conjuntos de datos tan grandes que aplicaciones informáticas tradicionales de procesamiento de datos no son suficientes para tratar con ellos

- **Bluetooth :** especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz.

- **Beacon:** Dispositivo de bajo consumo que emite una señal broadcast, y son suficientemente pequeños para fijarse en una pared o mostradores. Utiliza conexión bluetooth de bajo consumo (BLE).

- **C:** Lenguaje de programación creado en 1972.

- **C++:** Lenguaje de programación creado como evolución de lenguaje C, contando con características de programación orientada a objetos.

- **Cloud Computing:** Paradigma que permite ofrecer servicios de computación a través de una red, generalmente internet.

- **ECMAScript:** Especificación de lenguaje de programación publicada por ECMA International.

- **Event Loop:** programa que espera la llegada de un evento o señal para realizar una acción.

- **Framework:** Estructura conceptual que provee a un software de funcionalidades. Formado habitualmente por módulos o librerías.

- **Front-End:** Parte del software que interactúa con los usuarios.

- **Hardware:** Partes tangibles de un sistema informático.

- **iOS:** Sistema operativo creado por Apple para dispositivos móviles.

- **Java:** Lenguaje de programación orientado a objetos.

- **Javascript:** Lenguaje de programación interpretado, dialexto estándar de ECMAScript. Orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

- **Librería:** Conjunto de implementaciones con una interfaz definida. El término apropiado es biblioteca.

- **Major Number:** Identifica el driver de un dispositivo.

- **MacOs:** Sistema operativo creado por Apple.

- **Minor Number:** Identifica un dispositivo en particular.

- **Software:** Partes intangibles de un sistema informático.

- **Smartphone:** Teléfono móvil con mayores características y funcionalidades que un teléfono móvil tradicional, con poder de computación similar al de un ordenador..

- **Xcode:** es un entorno de desarrollo integrado (IDE, en sus siglas en inglés) para macOS que contiene un conjunto de herramientas creadas por Appledestinadas al desarrollo de software para macOS, iOS, watchOS y tvOS

- **Wifi:** Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.